NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A MODULAR APPROACH TO TIME-BASED UAN SIMULATION DEVELOPMENT

by

Richard Betancourt

September 2007

Thesis Advisor: Geoffrey Xie
Second Reader: John Gibson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE A Modular Approach to Time-Based UAN Simulation Development	**5. FUNDING NUMBERS**
6. AUTHOR(S) Betancourt, Richard	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The necessity to project naval combat power throughout the littorals has resulted in the explosion of growth in the development and implementation of wireless underwater networks. Contrary to the terrestrial wireless signal, which uses electromagnetic (radio) signals as a medium for the transfer of data, an underwater network utilizes acoustic signals to carry data. Additionally, unlike the terrestrial counterpart, the underwater acoustic network operates in a dynamic, ever changing environment that is susceptible to dramatic shifts in ocean water columns that are influenced by numerous parameters, e.g., density, temperature, depth, and current. Couple this with the mechanical impediments of electronic equipment, operating in a waterborne environment, and the problems begin to multiply exponentially. This thesis presents a new, standardized application programming interface for the development of acoustic physics models and network protocol stacks that can be dynamically loaded into an underwater acoustic network simulator. The interface will meet the needs of the United States Navy, scientific organizations, and private parties, by providing a key building block of a robust, modular based simulation framework that will allow rapid and cost saving research and development and testing of underwater networking technologies.

14. SUBJECT TERMS Underwater Acoustic Networks, Time-Based Simulation, Delay Tolerant Networks, High Latency Protocols	**15. NUMBER OF PAGES** 133
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A MODULAR APPROACH TO TIME-BASED UAN SIMULATION DEVELOPMENT

Richard Betancourt
Lieutenant, United States Navy
B.S., San Diego State University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Richard Betancourt

Approved by: Geoffrey Xie
Thesis Advisor

John Gibson
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The necessity to project naval combat power throughout the littorals has resulted in the explosion of growth in the development and implementation of wireless underwater networks. Contrary to the terrestrial wireless signal, which uses electromagnetic (radio) signals as a medium for the transfer of data, an underwater network utilizes acoustic signals to carry data. Additionally, unlike the terrestrial counterpart, the underwater acoustic network operates in a dynamic, ever changing environment that is susceptible to dramatic shifts in ocean water columns that are influenced by numerous parameters, e.g., density, temperature, depth, and current. Couple this with the mechanical impediments of electronic equipment, operating in a waterborne environment, and the problems begin to multiply exponentially. This thesis presents a new, standardized application programming interface for the development of acoustic physics models and network protocol stacks that can be dynamically loaded into an underwater acoustic network simulator. The interface will meet the needs of the United States Navy, scientific organizations, and private parties, by providing a key building block of a robust, modular based simulation framework that will allow rapid and cost saving research and development and testing of underwater networking technologies.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. UNDERWATER ACOUSTIC NETWORKING

The necessity to project naval combat power throughout the littorals has resulted in the explosion of growth in the development and implementation of wireless underwater networks. Contrary to the terrestrial wireless signal, which uses electromagnetic (radio) signals as a medium for the transfer of data, an underwater network utilizes acoustic signals to carry data. Additionally, unlike the terrestrial counterpart, the underwater acoustic network operates in a dynamic, ever changing environment that is susceptible to dramatic shifts in ocean water columns that are influenced by numerous parameters of which density, temperature, depth, and current are only a few. Couple this with the mechanical impediments of electronic equipment, operating in a waterborne environment, and the problems begin to multiply exponentially. This thesis will propose a new, standardized application protocol interface for the development of acoustic physics models and network protocol stacks that can be dynamically loaded into an underwater acoustic network simulator. This will meet the needs of the United States Navy, scientific organizations, and private parties, by providing a robust, modular based simulation framework that will allow rapid and cost saving research and development and deployment of underwater networking technologies.

Due to the physical characteristics of the ocean, acoustic networks are currently the only viable means of

communicating wirelessly over distances of more than a few tens of meters [1]. Radio waves will travel long distances in water but only if the sea is conductive and the frequency of the radio signal is between 30 and 300 Hz. The equipment that this requires is prohibitive for relatively small autonomous underwater vehicles (AUVs). For example, while testing of large vehicles, up to 72 inches and larger in diameter, is underway, the Navy currently uses vehicles that are no larger than a man's leg in actual mine clearing operations off the coast of Iraq [8]. Optical forms of wireless communication are too unstable in an underwater environment where, due to wave motion and other biological obstacle, the refraction and deflection of light are too unpredictable and prohibitive for use in communications.

While numerous organizations would benefit from and are requesting the development of AUV based networks, the costs associated with real world testing of network protocols is prohibitive. For this reason, many research institutions are looking towards simulations to provide some clues as too the performance characteristics of the developed technologies. This greatly decreases the time and overall cost of research and development. However, current methods used by the networking community to develop underwater networking protocols are limited to event-based simulation and do not support realistic, time-based simulation. While event-based simulation is adequate in environments where the latency can be considered non-existent when compared to the processing/user response time, in the ocean environment the latency must be considered. All the major network simulators are based on well known terrestrial networking hardware and protocols. For this reason, most underwater

acoustic network protocols are evaluated using modified parameters of the radio-based models. However, they still lack the ability to accurately model the physical medium – the ocean.

B. UNDERWATER ACOUSTIC NETWORK DEVELOPMENT METHODOLOGY

This thesis will describe a software architecture that will lend itself to the testing of UAN protocols. What is needed in the world of network simulation tools is a robust simulation base that will allow for the dynamic loading and updating of a model of the physical medium and its key characteristics as well as the network nodes' communication protocols. Depending on the acoustic model implemented, users should be able to modify key parameters, e.g., temperature, salinity, or background noise. Developers and users should also be able to develop acoustic models in programming languages that are: 1) more popular in the scientific communities, e.g., C or Fortran, and 2) can more efficiently use the host system's resources. These models will then be loaded, at run time, through the use of the Java Native Interface API and an associated wrapper class. The client node simulation environment will also allow for the dynamic loading of networking protocol stacks specifically designed for the ocean environment that need to be evaluated. Lastly, the simulation architecture will allow users to develop and implement a node location simulation that will model node movement in the environment.

This simulation base will build on a previous thesis by Lieutenant Brian Long which provided a simple framework for a distributed client-server based simulation and provided a proof-of-concept of a distributed network simulation [3].

By applying the modular characteristics briefly described above, and detailed in the following chapters, and with the distributed framework developed by Lieutenant Long, the attractiveness and feasibility of using COTS equipment for testing, and deployment of this system to actual Navy and civilian units, becomes a very feasible possibility.

C. THESIS OUTLINE

The thesis is organized as follows. Chapter II will provide some background information into the nature of the underwater acoustic environment. Chapter III will discuss the modular application interface of the central host, or server. Chapter IV will detail the application interface of the network nodes, or clients. Chapter V will show the implementation of a simplified acoustic model, location model, and a UAN protocol stack. The final chapter, Chapter VI, will close with ideas on possible future avenues of development.

II. BACKGROUND

A. OCEAN ENVIRONMENT

We will start with an introduction into the ocean environment, the desire for organizations to deploy autonomous networks into the vast seas, the nature of sound in the ocean, and why this must be the medium of communications. This first section will only briefly touch on the more important considerations of acoustic networking and the physics involved. The reader is encouraged to refer to the references, given at the end of this thesis, as a starting point for more detailed discussions relating to this area of computer science and physics.

The earth's oceans comprise a greater proportion of the total surface area than does the land. The potential uses of these large volumes of water are being weighed heavily by nations and private companies.

The trade between nations relies on the ability to safely navigate across the vast stretches of water and narrow straits. Potential biological and geological discoveries still wait in many previously unexplored areas. The world's oceans also provide a natural defensive buffer along the coasts of many nations, as well as a volume of space in which to conduct covert offensive operations [1][8][11]. Enabling technologies that would allow each of these to continue or improve include the deployment of unmanned autonomous underwater vehicles (AUVs) and the acoustic networking technologies that allow for underwater communications.

The oceans are harsh and dynamic environments. Temperatures can vary greatly depending on the location and depth. The depths of the oceans may be shallow, tens of meters, or deep, the Mariana's Trench is just over 10 kilometers in depth. The associated pressures that are encountered as one proceeds deep make manned missions to the depths limited and expensive. And, depending on the sea state, the ability to conduct manned operations on or just below the surface can be treacherous. For these reasons, AUVs provide the right platform for safe and continuous operations in the world's oceans.

As the AUVs perform their missions they will be collecting data. The AUVs may be deployed as a single unit or as part of a larger cluster of vehicles. In either case, a means of communicating between AUV nodes or from AUVs to a surface/subsurface vessel will be essential for mission accomplishment.

Terrestrial based means of communication do not perform as desired or cannot be utilized without creating additional problems. Wired, or tethered, links to the AUVs are simple and easy to implement. However, this has several drawbacks. First, maintaining the tethered link requires some kind of mother ship to remain on station. Second, the range is limited to the amount of wire on board the AUV, the mother ship, or both. Third, the link connecting the AUV and mother ship restricts maneuverability of both vessels and could potentially foul either vessel's propeller.

Radio communications are only effective for short distances for medium to high frequencies and for long distances at very low frequencies. The proper reception of

such longer distance radio signals would also require a large antenna, potentially limiting an AUVs payload capability and maneuvering ability while only supporting extremely low data rates. Optical forms of communication generally fail beyond a few tens of meters due to the refraction of light [1].

Acoustic signals provide the optimal solution. By utilizing acoustic signals we gain the benefits of wireless communication without the distance limitations or cumbersome antenna. However, the properties of sound in water do not allow performance comparable to terrestrial radio network communications. The speed of sound and the path the acoustic signal take are dependant on many variables: temperature, depth, frequency, biologics, etc [10]. The average speed of sound varies but can usually be considered 1500 m/s [9]. This speed limitation, along with other phenomena, such as shadow zones, channel ducts and very limited bandwidth, present unique challenges for network engineers.

B. DEVELOPING FOR THIS ENVIRONMENT

The greatest challenge facing protocol designers is to find an environment in which to test the underwater networking protocols. As mentioned above, the ocean environment presents challenges to traditional wireless networking protocols. The speed limitations of an acoustic signal and wave nature of the acoustic path require delay and fault tolerant protocols.

A tank, pool, or lake does not provide a realistic testing environment. The ideal testing environment would be

at sea with real AUVs. However, the cost associated with performing this kind of testing is prohibitive. Arranging at sea training requires coordination between all interested parties. The man-hours associated with preparing the ship, AUVs, associated equipment, and actual deployment and testing are tremendous. This is true especially if the desired testing environment requires extensive travel time. And, all the preparation, detailed testing plans, can go to waste if Mother Nature does not cooperate.

Another method would be to test protocols in commercial network simulation packages. These greatly reduce the costs associated with performing the protocol testing and minimize the development time. However, the majority of network simulation packages are event based and cannot model the time varying ocean environment well, if at all. Secondly, these simulation packages were designed to test terrestrial networks that model the wired and radio mediums used for networking. In order to use these simulation packages, the users must make some assumptions and modifications to the existing radio wireless models which will have an effect on test results [12].

C. A DISTRIBUTED TIME-BASED SIMULATION

Recent work in the area of underwater acoustic networks at the Naval Postgraduate School showed the feasibility and application of a distributed, client-server based architecture for network simulation. Lieutenant Brian Long developed a simulation framework that allowed clients, network nodes running on commercial-off-the-shelf (COTS) computers, to connect to a central server [3]. This configuration has several advantages, discussed below.

A distributed simulation can be used to allow the sharing of the load. The work associated with running and evaluating UAN protocol stacks was pushed out to client computers while the server was left to handle the bulk, and perhaps the most important part of the simulation: the calculation of the acoustic signal properties. The distributed nature of the framework will also enable the more accurate development of a time-based simulation environment. The time-based simulation will allow the ocean environment to be modeled realistically; the numerous parameters involved with determining the acoustic signal path and properties can be taken into account rather than leaving it to a quasi-random process of discrete event based simulations, which most terrestrial network simulations are.

However, what is lacking from the framework developed by Lieutenant Long is an application programming interface (API) that would allow for the dynamic loading of network protocol stacks and acoustic models. This thesis will describe an API which will allow developers and users to easily create, test, and use existing and custom built models and protocol stacks without the requirement of learning a new programming language. This will greatly improve development time and provide more realistic test results. Development time is greatly reduced since no new programming language is required to be learned, errors associated with porting code are negated, and users can quickly migrate the code directly from testing to deployment. By utilizing more accurate acoustic models, users can have greater confidence in the results they receive from the simulation.

D. REQUIREMENTS OF THE TIME-BASED UAN SIMULATION

This section will outline the general requirements of the overall system. Chapters III and IV will detail the requirements, development, and usage of the modular server and client, respectively. Some of these requirements were originally outlined in [3] and are reiterated here for consistency.

The core of the simulation system should be designed so that it could be run on various types of hardware; a dedicated server farm running complex models or laptop computers by users in the field who need to quickly predict network performance for the deployed area using real-time environmental data, which could change in as little as a matter of hours. The system is designed to simulate the underwater environment in as many aspects as necessary and consistent with the goals of the developer. This not only requires calculating the time delay in propagating an acoustic signal from a source to a destination, the latency, but the effects of the environment upon the acoustic signal itself, i.e., frequency effects and power attenuation. In order to perform these calculations upon the acoustic signal, the parameters affecting the water column should be mutable. These parameters could be: salinity, temperature (uniform temperature applied across an entire volume or as a more complex sound speed profile type of gradient), biologics (physical density, number and type per square mile, noise level), shipping/mechanical noise (in harbors, shipping lanes, and, more generally, the littorals, which can greatly affect performance of acoustic signals), sea state, and bottom type.

The acoustic model may also take into account the wave nature of sound. This applies specifically to the phenomenon known as shadow zones. A client UAN node may be within a reasonable range of the source but due to the wave nature of sound and the physical properties of the water column, the sound signal will not reach it.

Most of the computational acoustic models developed and being developed in the field of ocean acoustic modeling are written in languages such as C and Fortran [9] [13]. These languages have been adopted by the scientific communities because of the ability to interact with the hardware at a low level as well as the speed and optimizations that these languages allow. Interpreted languages or programs that require the use of a virtual machine may not be able to duplicate these capabilities [17]. For realistic modeling, complex mathematical models have been developed and as such require great amounts of computational power. Being able to run pre-built and optimized code would improve overall system performance and ease development requirements as porting of proven, reliable code would not be necessary and the benefits of software reuse could be maximized. Development time would also improve since learning a new language would not be required. While learning a new programming language may not be difficult, learning a programming language well enough to exploit the hardware platform for maximum benefit can take time even for seasoned programmers. A large selection of the various models available can be found at the Ocean Acoustic Library [13].

The simulation system should also be able to run on a single host or distributed across a network. This has

several benefits which will allow this system to be easily utilized by deployed units and larger corporate type research departments. Depending on costs and/or currently installed infrastructure, the system can be run on a single hardware platform, e.g., laptop or be set up on multiple hosts to provide researchers, who may be across campus or across the country, to connect and test acoustic models and protocols.

The simulated UAN node, or client in the distributed aspect of this system, should also implement a modular architecture. The node, as in the server acoustic model, should allow for the loading of protocol code that was written in other languages. This would hold the same benefits as mentioned above. Network parameters should be dynamically adjustable. Client users should be able to change the frequency, power, send and receive capabilities (half-duplex vs. full-duplex), as well as other properties of the acoustic modem/physical layer. The modeled node should be maneuverable, in that it should be able to change depth, speed, and course depending on the type of network being modeled. To help in the acoustic simulation, the server should maintain the location of each node in the simulated environment.

E. SURVEY OF CURRENT UAN SIMULATION METHODS

This section will briefly outline some of the current methods being employed by professionals working in the UAN field; focusing on how the underwater environment is simulated and development of protocols. While not all inclusive, many aspects, pros and cons, will be discussed

and the benefits of the proposed framework introduced in this thesis will be mentioned.

The Optimized Network Evaluation Tool (OPNET), by OPNET Technologies, is a powerful discrete event network simulation tool. It is designed to allow networking professionals to develop and test protocols and other network technologies. The OPNET Wireless Modeler allows network engineers to model, simulate, and analyze wireless networks. Additionally, grid computing capabilities are built into OPNET to allow for distributed simulation. An extremely powerful and popular network simulation tool, OPNET has been used by many scientists working in the UAN field [5] [12] [14] [15]. However, OPNET is not an ideal solution for UAN simulation. As pointed out in [15] and [16], OPNET does not model the physical, underwater medium. Also, the OPNET Wireless Modeler does not have an acoustic modeler; the acoustic signal is based on, with some modifications, the provided radio models in OPNET [5] [12].

The Autonomous Undersea Systems Institute (AUSI) has developed a distributed, client-server based, simulation environment [6] [7]. The Cooperative AUV Development Concept (CADCON) was designed to primarily evaluate the interaction of existing autonomous vehicles in an underwater environment. CADCON offers developers a simplified ocean environment simulation which can be configured via configuration files prior to execution. This underwater network simulation tool provides a more realistic simulation of the physical medium. However, if scientists desire to use a different, more accurate or detailed environment simulation, then the CADCON source code would have to be

modified and recompiled. For example, the CADCON environment simulation does not model the noise generated by shipping/pleasure craft or wave motion. Another drawback for using CADCON, specifically in the research/academic environment, is the inability to easily plug in various protocols. As it currently stands, CADCON simulation environment is designed to work with existing vehicles, via socket communication, with their existing hardware/software. If it was desired to have a different protocol used, the vehicle would need to have the correct software loaded. AUSI is working with the University of New Hampshire to develop an ability to plug in various protocols and run them on a simulated vehicle via VMware.

This thesis will outline an UAN simulation framework that will address the weaknesses of the above mentioned simulation tools. First, the framework proposed here will allow various types of ocean environment acoustic models, written in Java, C/C++, or Fortran, to be easily plugged into the simulator and used with little or no source code rework. The framework will implement a client-server based environment to allow for load sharing and simulation distribution. Lastly, in a similar manner to the ocean environment acoustic model, existing or experimental protocols may be plugged into the client side node with ease.

The following chapter will detail the server requirements and Chapter IV will discuss the simulated UAN node or client requirements of the UAN simulation framework.

III. THE SERVER

A. SERVER REQUIREMENTS

The previous chapter provided a general description of the UAN simulation server. In this chapter we detail the server requirements, develop use cases, and then describe the modular server architecture. To reiterate, the server's main purpose is to serve as the simulated communications medium, the ocean, and "broadcast" simulated acoustic messages to connected nodes. While this is the main focus of the simulation, the details of all the associated classes that make up the server will not be detailed. Those classes handle the basics of the client-server networking and the user interface. It is assumed that the reader is familiar with this area of programming and the details of how they are implemented are not the focus of this thesis and no new insights may be gained by investigating those areas. However, source code and UML class diagrams will be given in the appendix.

1. Working with Acoustic Models

In Chapter II we noted that the acoustic model programs are mainly written in C and Fortran. Although there is much debate about the progress Java has made in recent years in terms of performance, these languages still have their benefits. Being able to generate object code in the native form of the host computer will always have greater performance characteristics if not during run time, then during initial startup and overall memory usage. Secondly,

these are popular languages that the scientific community has been using for years and they have developed high performance libraries and tools. The Java community has just recently been developing numerical libraries.

This thesis does not argue either for or against the computational capabilities of Java or C, but must take these facts into account and is one of the major requirements for the UAN simulation system. The UAN simulation should allow for the seamless utilization of acoustic models, potentially written in a language other than Java, with little to no code modification by the users of either the UAN system or the writers of the acoustic modeling software. Given an "acoustic signal", a source, and a destination point, the acoustic modeling software should return an "acoustic signal" appropriately modified to reflect how the destination point will hear it/receive it.

The acoustic modeling software may also allow users, during runtime, to modify some of the physical parameters of the ocean environment, i.e., temperature, salinity, background noise level, sea state (to name a few). Due to the ever changing and unpredictable nature of the ocean environment, allowing deployed units to easily, at runtime, modify the characteristics of the simulated environment to reflect those that are actually found at the desired location could allow them to then make needed modifications to the network topology to better ensure end-to-end connectivity. In this case, the simulation is a tool for deployment management rather than protocol or application development.

2. Simulating Client Locations

The server should at all times maintain a list of client nodes and their location in the simulated ocean environment for use in determining broadcast recipients of an acoustic signal. The location of the node must be in three dimensions. Simulated client nodes may be stationary or following a specified course. The course can be changed by the client at runtime. This frees the client nodes from the burden of determining who is capable of receiving signals as well as mitigating potential timing skew between nodes.

This portion of the UAN simulation differs from that developed by Long. The server does not maintain a complete “god’s eye view” of the network topology. No XML file, with node locations and channels is used. The use of an XML file that statically predetermined the number of nodes and their locations is too limiting. A node may join the simulation at any time and is not predetermined by the choice of XML file the server has loaded. Also, the channel on which a node is listening should not be fixed nor is it required to be known by the server. The channel a node is on is known by the node and, depending on the protocol, any other nodes in range. A node may also change its operating channel or wish to listen to all channels that it physically can receive.

These are the main areas of the modular UAN server that need to be covered here. Other aspects of the server have been designed with modularity as a guiding principle, but are not discussed. One of the software components not discussed in the body of this thesis is the user interface

component. Depending on how the UAN simulation is being run and on what type of system, the user interface component will handle the sending and receiving of information to and from the system and present that information to the user in the desired format, i.e., console based (text) or through some kind of GUI.

The following section will look at who the actors are of this portion of the system and what functions they need to perform.

B. USE CASE ANALYSIS

In the previous section the word “system” has been used rather freely but will now be defined more concretely for purposes of performing use cases analysis. The system will be comprised of the Simulation Control, Network Manager, and User Interface. The Simulation Control portion of the system provides the main logic and oversight of all actions that take place; it is the hub of the overall UAN simulation. The Network Manager performs the required network socket instantiation for all incoming client requests. The User Interface will provide an abstraction layer for the sending and receiving of information between the user and the simulation. Together, these three parts will comprise the system.

This section will examine those actors that will receive some sort of stimulus from the system, make a request to the system, or both. The actors may be software components that are vital to the overall performance of the simulation or the simulation users. The acoustic model and

location simulation plug-ins are examples of the former. We will examine what the actors are and the interaction between these actors below.

1. The Acoustic Model

The first entity we will look at will be the Acoustic Model. Despite being the core, or heart of server, the Acoustic Model must be considered a separate and autonomous actor. This is important for a couple of reasons. First, it allows for the complete abstraction of the acoustic model implementation. The acoustic model used may be simple or complex. Spherical or cylindrical spreading calculation of attenuation only is a simple mathematical model that only requires a couple of parameters and requires only a single line of code. However, a more complex model may be implemented that will dynamically model an ocean environment in real time and require the use of a multiprocessor system or server farm, for example. Second, facilitating the first, it removes dependency on what programming language is used for implementing the acoustic model. This is an important consideration when working with physicists in the acoustic modeling arena. Requiring subject matter experts to become familiar with a particular language can limit the pervasiveness of the model framework. Therefore, it is necessary to consider the Acoustic Model as a separate actor.

As a separate, autonomous actor, the Acoustic Model must perform, at a minimum, one task, or function: given an acoustic signal, determine the acoustic signal properties at a specified destination location from a given source location. This function will be requested by the System.

However, limiting the functions that an Acoustic Model can implement is short sighted and doesn't allow for greater simulation control. For example, deployed units may wish to alter the ocean environment parameters dynamically before and during the simulation to allow for a more accurate simulation of the actual environment. This can be accomplished by specifying a few more functions that an Acoustic Model may implement. The user should be able to query the Acoustic Model for a list of dynamically adjustable parameters. The user may request a list of all the parameters and their current values, alter their values, if desired, and submit them to the model. The Acoustic Model, given an allowable parameter and value pair, will dynamically set the parameter to the given value and all future acoustic modeling calculations will use that value.

Figure 1, below, diagrams the functions which users, via the System, and the System should be able to request of the Acoustic Model. The figure also clearly shows that the Acoustic Model could actually be considered a separate system. This is a main point in the development of the overall UAN simulation. Specifically, the Acoustic Model, as a separate system, could be running on the same system as the bulk of the simulation software or on another, connected platform or platforms, i.e., server farm, or grid/cluster computers. This is similar in the approach that was taken in [6] and [7]; however, as will be discussed in Chapter V, the framework introduced here is more flexible; the implementation of the model can be cleanly abstracted.

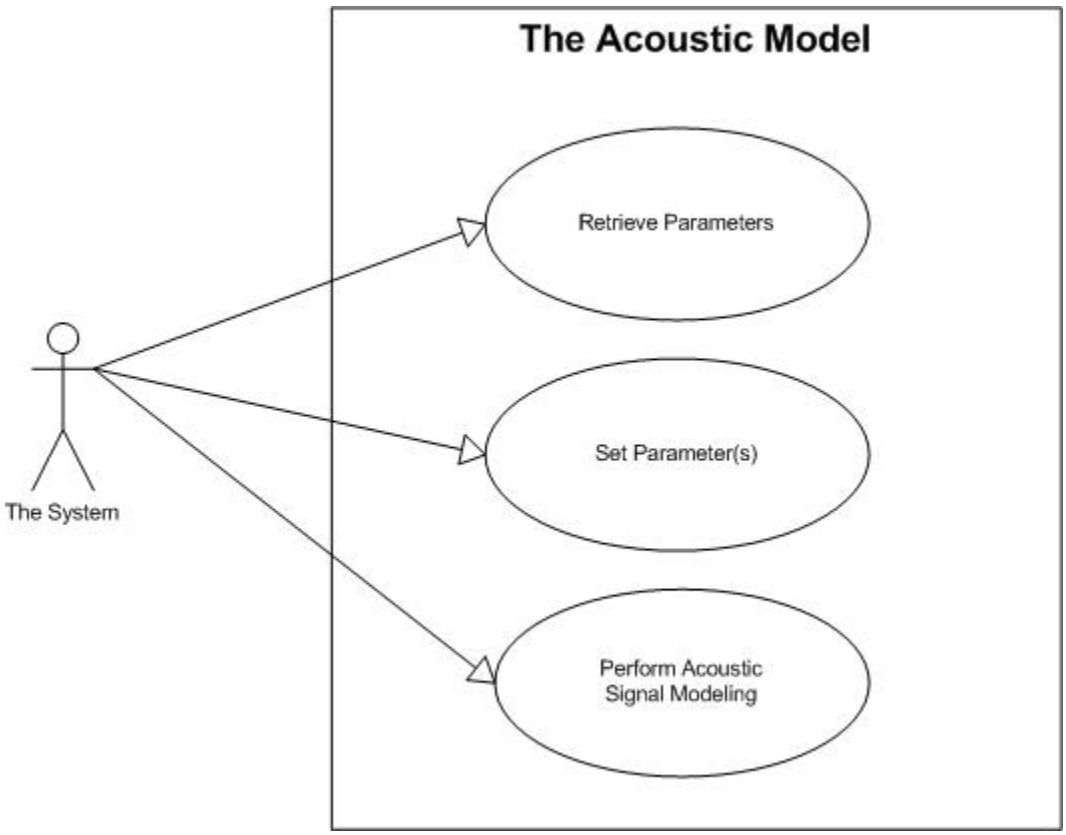

Figure 1: Acoustic Model and System Use Case.

2. The Location Simulator

The next actor we need to consider is the Location Simulator. The purpose of this actor is to keep track of all the simulated node locations. This may be a simple static model where the nodes are “anchored” to the ocean floor. Or, the nodes could be a type of unmanned underwater vehicle (UUV) that can change course and speed throughout the simulation.

Figure 2 shows the functions that the Location Simulator should handle. The Location Simulator will, similar to the Acoustic Model, be an autonomous component

that will serve the System. The majority of requests that are made to the Location Simulator will come externally from the clients that will modify the simulated node's location. The System should be able to perform the following functions:

- Add a node to the Location Simulation
- Set a node's location in three dimensions: latitude, longitude, and depth
- Set a nodes speed and course
- Retrieve any of a given node's parameters: latitude, longitude, depth, speed, and course.

Again, by making the location simulator a separate entity we can abstract the implementation.

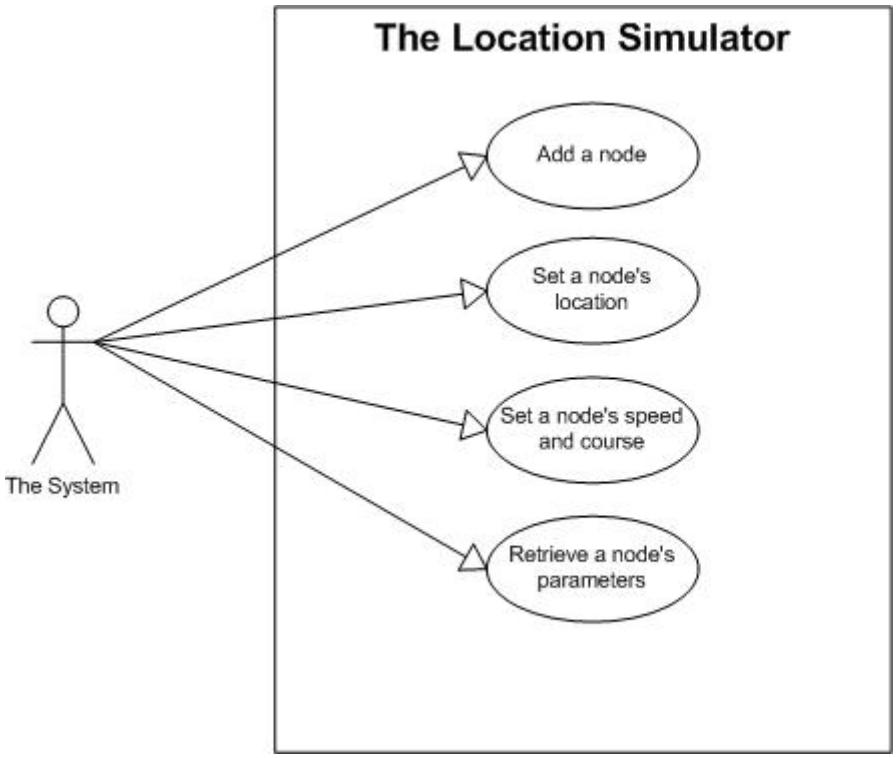

Figure 2: Location Simulator and System Use Case.

3. The Server-Side User

The server-side user is the next entity considered. We distinguish this user from the client or host node, which will in most cases have a different user. The client-side user will be discussed in Chapter IV. However, for now we can simply and clearly state that the server-side user will control the administration of the UAN simulation and the client-side user will be administering the running of the simulated client node wherever that host computer or hardware device is located on the connected network. The administrative capabilities that the server-side user should control the basic server settings: starting, stopping, setting network connection parameters. This user will also have the following abilities: (1) select desired Acoustic Model, (2) select desired Location Simulation, (3) set Acoustic Model parameters, (4) view Acoustic Model parameters, or (5) view an individual node or a list of all connected nodes.

Figure 3 shows the functions that the server-side user can access on the System. The functions that require interaction with the Acoustic Model are shown as part of the System vice as the server-side user interacting directly with the Acoustic Model. There are a couple reasons for this. First, the user interface, as mentioned above, is part of the System and is the means of the server-side user communicating with the server. Secondly, this centralizes the software interfaces that would be required to only one, vice two or more, and will improve software maintainability and development.

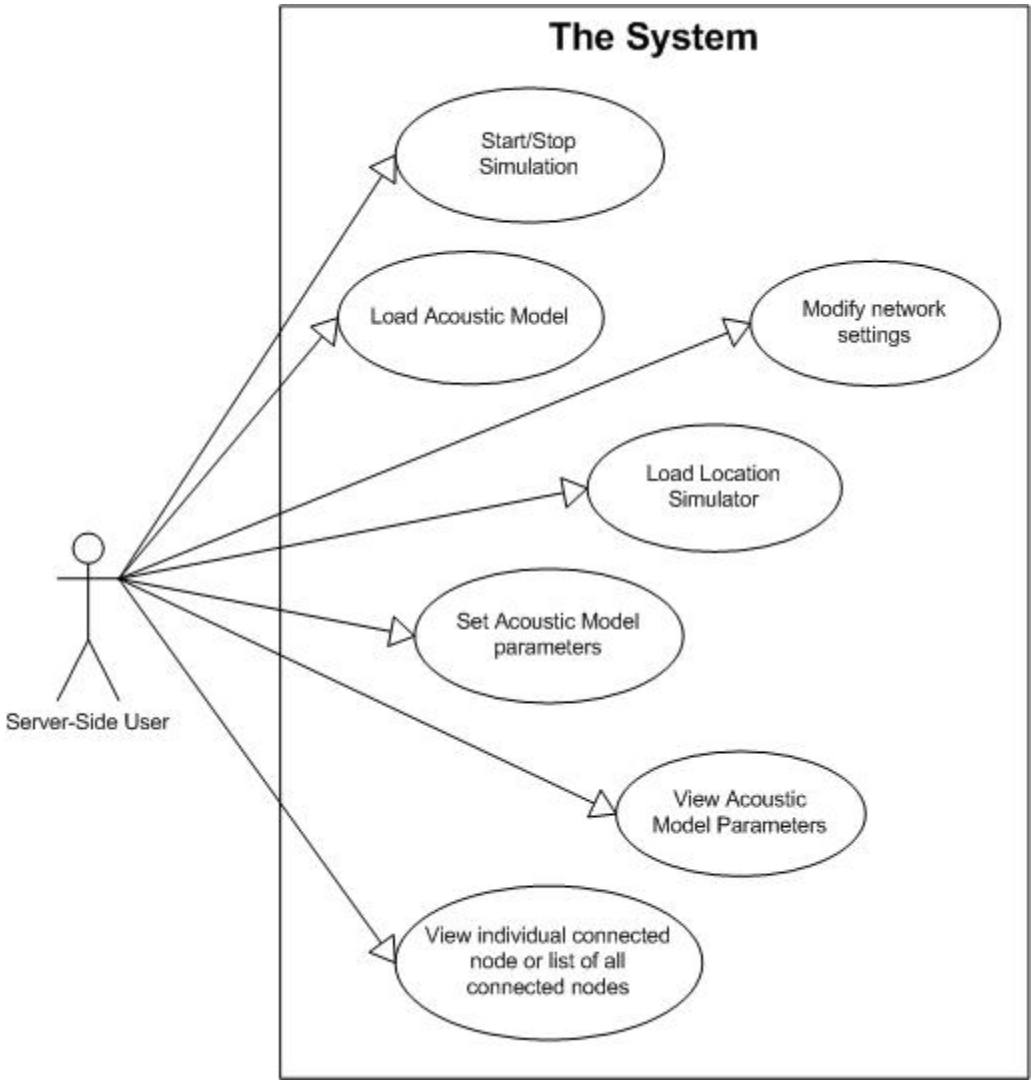

Figure 3: System and Server-Side User Use Case.

4. The Client

The next and probably most important actor of the system will be the network client or simulated node. This client should not be confused with the client-side user. Here the client is another computer or hardware device simulating the underwater node that requests the simulation services of the UAN Simulation server, while the client-side

user, as will be further discussed in Chapter IV, will administer the node simulation on the connected host platform.

Clients request the system to handle the simulated broadcasting of an acoustic signal through the medium of the underwater environment. These clients may be running on the same host system as the server, across a small LAN setup in a networking lab, setup across a routed LAN such as between the physics department and computer science department, or across the country between Universities or other agencies (WAN). The actions required between the client and the server can be broken into administration and simulation based. The administration functions that the system must be able to perform for the clients include the establishing and managing the connection to the server. The simulation based functions that the system must handle for the client are: broadcast acoustic signal and update node course, speed, depth, location information. Figure 4 provides a visual representation of the functions the System must perform for the client.

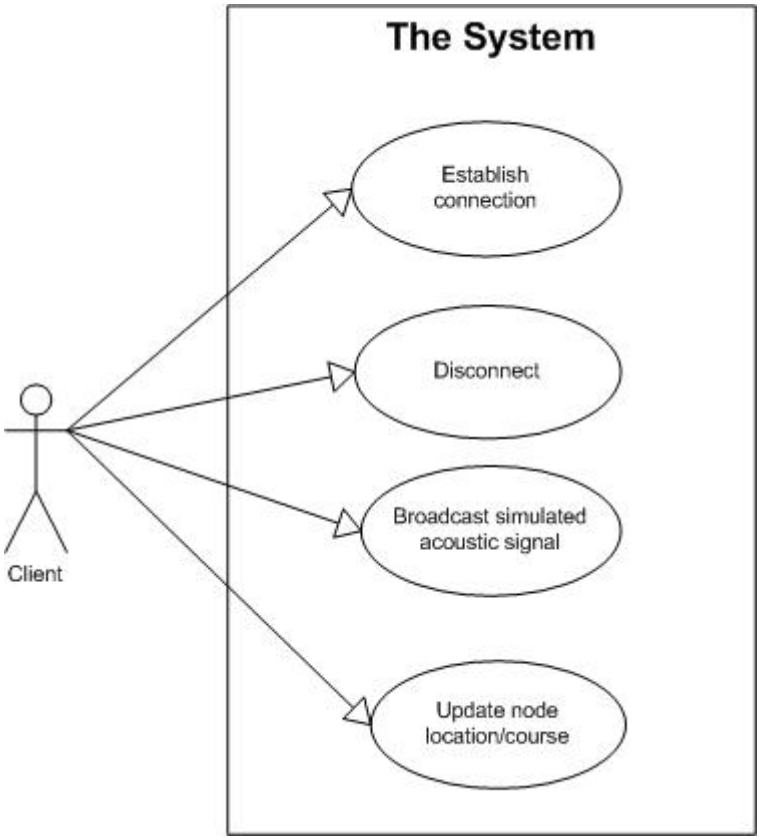

Figure 4: System and Client Use Case.

This chapter outlined the actors, or entities, of the server and showed the more essential functions that the server must be able to satisfy. The following chapter will discuss the client.

IV. THE CLIENT

A. CLIENT REQUIREMENTS

The client is responsible for implementing some type of network protocol stack to allow for communications between nodes in an underwater environment. The protocols necessary for working in this high latency, error prone, highly variable, wireless networking environment are constantly being worked on. However, the means of evaluating the protocol performance is not ideal. When resources, timing, and weather all work in your favor, then an actual test at sea may be possible. Yet, the coordination required to accomplish at sea testing is enormous and you cannot always count on Mother Nature to cooperate.

Other methods used to evaluate the protocols are ad hoc, at best. They rely on networking tools designed to test out terrestrial networking technologies and protocols. In order to test out protocols designed for the harsh ocean environment, they use the provided wireless (radio) mediums with some slight adjustments meant to model the delays and variability of the ocean environment. Despite the similarities in the wave propagation of the signal the other variables of the ocean are not modeled to a fine enough degree with those tools and the insights gained in to the UAN performance are questionable. The current methods used and the validity of the results obtained are discussed in [15].

The client must be able to allow users to use various types of protocols without having to worry about the details

of the simulated medium used to broadcast the signal. This thesis builds on the previous work by Brian Long, which abstracted the protocol stack to allow protocols to be easily evaluated.

The client user needs a host system in which they are able to plug in various types of protocol stacks, dynamically, and with no recompiling of the client source code. This should provide benefits, similar to what we saw for the server, to the client user. The first will be to allow users to use existing protocol stack code, written in C/C++, Fortran, or Java, with little or no modification of the original protocol or client node source code. This will greatly improve the development and testing times. Interested users and developers do not need to learn a new programming language to use the system. There is no need to translate the code from one language to another, which could introduce numerous software bugs; no time wasted while trying to learn a new programming language. Automated tools are available to convert from Fortran to C, for example, but the self-documenting nature of the original program's comments could be lost. Another benefit would be that the original performance gains that were programmed into the original code may be lost when translated, either by hand or with the automated tools. The basic client requirements will be stated below.

The most important requirement of the client portion of the simulation is to allow users to quickly and easily plug in various types of protocol stacks dynamically. Client users should have little to no source code rewriting or recompilation when attempting to integrate their protocol

stacks into the simulation. Secondly, the clients should be able to make use of the simulation server whether they are running on the same host system as the server, across the room or campus, or across the country. This will provide the abstraction necessary for the protocol developers to concentrate more on protocol details rather than the details associated with the medium.

The following section will discuss the actors and associated functions that they will request/fulfill.

B. USE CASE ANALYSIS

The system will be compromised of the following components: Client Controller, Network Manager, and User Interface. The Client Controller provides the main logic and coordinating body for the client. All non-simulation related communications, i.e., administrative setup, will go from the Client Controller to the Network Manager and out to the server. The Network Manager is responsible for establishing, maintaining, sending, and receiving communication to and from the server. The User Interface will provide the controls necessary for the user to interact with the Client Controller as well as provide the client with a means of interacting with the simulation protocol stack. With the system thus defined, we can now detail the actors and their associated functions.

There are three actors that will require some form of interaction with the system and they are: the client-side user, the Protocol Stack, and the UAN Simulation Server. Each of these will be discussed below.

1. The Client-Side User

The first of these actors is the client-side user. The user will require the ability to change the protocol stack being used by the client node. The protocol stack the user may choose to run on the client may be written in various languages. The user may also desire to see the settings of the acoustic modem, i.e., frequency or channel being used, power settings, etc. The user may desire to dynamically alter the node's simulated course, speed, and depth during runtime. Conversely, the user may wish to leave the node in a static location or course.

The user will also require the ability to connect and disconnect from the simulation server. The user must also have some way of interacting with the network by sending and receiving communications to and from the simulated network. In other words, the user must have the ability to use the protocol stack to communicate with other nodes.

Figure 5 provides a visual representation of the functions the client-side user should be able to perform upon the System.

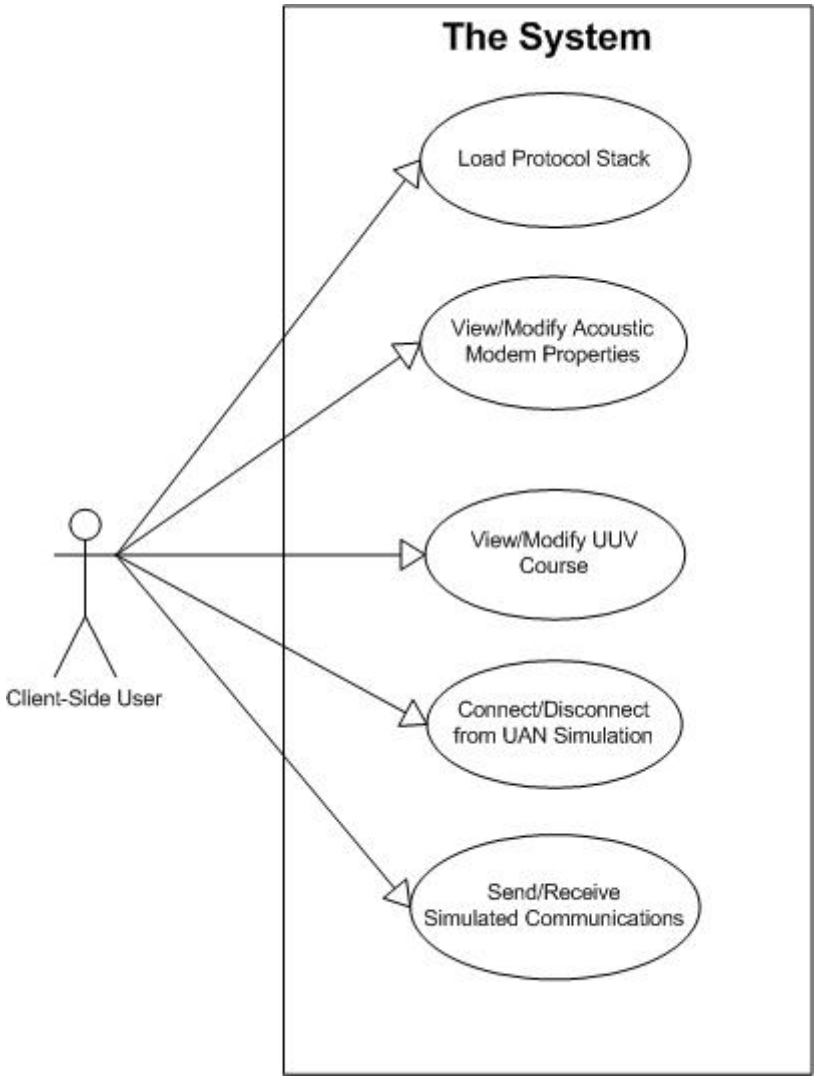

Figure 5: System and Client-Side User Use Case.

2. The Protocol Stack

The second entity that will require interaction with the system is the Protocol Stack. The Protocol Stack forms the most important part of the client as it is what carries the software code that needs to be evaluated in the underwater environment. The Protocol will receive input from the user or some automated program that the user has implemented. The input from the user will move through the

stack and will leave the stack at level 1, the physical layer (in the standard OSI model), and be sent through the simulated network.

The Protocol Stack will also simulate the features found at the hardware level, acoustic modem, and must, therefore, allow some means of the user altering those properties. That is, the Protocol Stack will contain the power, frequency, and other properties that are usually part of the acoustic modem. This will also allow the user to select the send and receive capabilities of the modem, if desired, i.e., half-duplex vice full-duplex.

Two diagrams are given here to show the functions that the client-side user may perform upon the Protocol Stack and the Protocol Stack upon the System; Figure 6 shows the client-side user functions upon the Protocol Stack while Figure 7 shows the Protocol Stack functions upon the System.

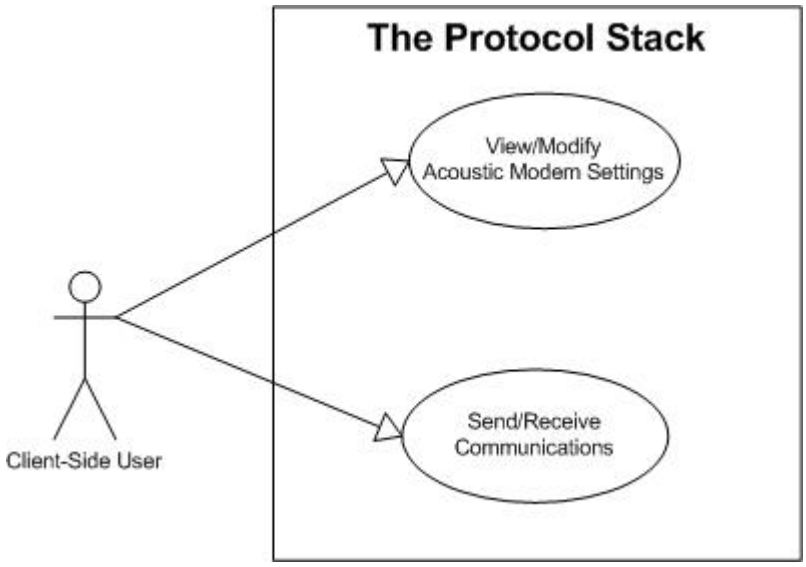

Figure 6: Protocol Stack and Client-Side User Use Case.

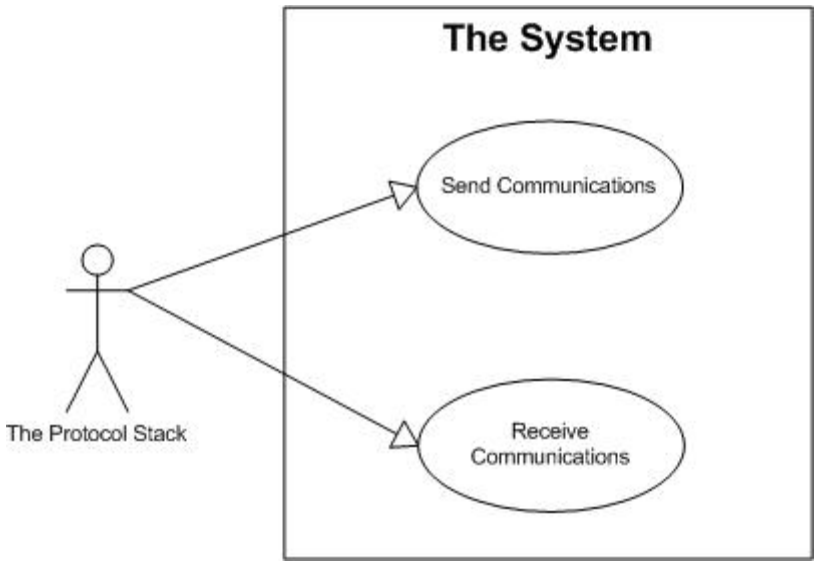

Figure 7: System and Protocol Stack Use Case.

3. The UAN Simulation Server

The final actor that must be considered is the UAN simulation server. The server will respond to incoming requests for connecting to, and disconnecting from, the simulated network. Simulated communications will be sent to the server and any incoming communications will be received by the system. These communications can be administrative, for example communications relating to connection establishment, or simulated acoustic network communications.

The handling of the delayed nature of the communications, the latency, and the more pronounced occurrence of collisions is an important topic and will be addressed here. This thesis builds on previous work and follows similar logic as found in [3]; the client-side simulated node is responsible for handling the latency and determining potential collisions. This is logical for a couple of reasons. One, this further frees up resources on

the server if it does not need to manage the timing of outgoing messages and determination of collisions for potentially many simulated clients. Second, the simulated nodes may have the capability to handle various communication schemes, i.e., half-duplex, full-duplex, or multi-channel simultaneous communications and it is logical to leave the handling of these schemes to the clients that are then free to implement as appropriate.

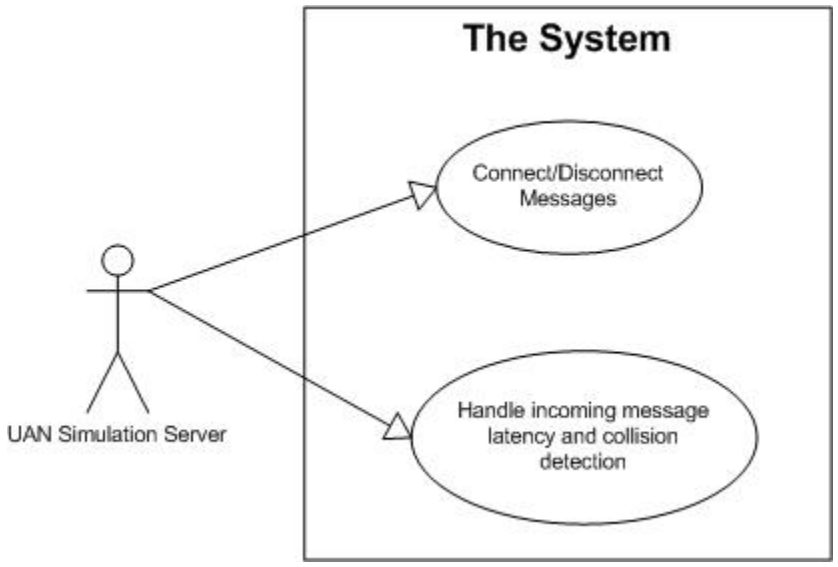

Figure 8: System and UAN Simulation Server Use Case.

This chapter discussed the entities and their associated functions on the client side of this distributed simulation. The following chapter will detail the implementation of these requirements into a modular software architecture.

V. THE MODULAR IMPLEMENTATION

A. USING JAVA

Java provides numerous advantages and is ideally suited for this type of application. The UAN simulation by design is distributed and should be able to run on multiple types of platforms. These may be small laptops, desktops, or powerful servers. Regardless of the hardware platform that the system is being run on, the simulation should perform seamlessly. And Java provides that power.

Another advantage of Java is the ability to make use of native methods, or code that is written in a different language, C for example, and compiled for a specific hardware platform. This will allow simulation users to make use of software that has been optimized for a specific purpose or platform. The acoustic models often require highly optimized code to perform the multitude of calculations that are necessary. Also, many of the acoustic models have existed for quite some time and have been written in older languages, such as Fortran. Rather than having to go through a painstaking and error prone process of translating the code, Java provides a means of calling those programs with little or no modification to the original source code.

In order to make use of the native code, Java uses what it calls the Java Native Interface (JNI). The JNI is a powerful addition to the Java API family and was introduced early by Sun Microsystems into the Java Development Kit (JDK) 1.1. A Java application that makes use of the JNI to

call native code will have several advantages that are not built into the Java Virtual Machine (JVM).

Native code has the ability to call and make use of specific hardware functionality which would otherwise not be accessible from the JVM. Source code may be compiled with specific optimization flags set that will create code that is ideally suited for a specific hardware platform, which would not be possible running only through the JVM. However, developing software that depends on the native methods has some drawbacks.

Programming with the JNI reintroduces many of the errors that Java was designed to eliminate. First, Java was designed to allow programs to be run on any hardware platform that had a supporting JVM. This “write once, run anywhere” development philosophy is destroyed when Java programs are written that require native methods. The native methods, if they are going to work on other platforms, must be recompiled for that specific platform. Secondly, memory management, or garbage collection, is not provided for the native written code. Native code has the ability to create and modify Java objects as well as call other Java methods that may create or return new instances of a Java object. This presents possible memory leaks if not properly handled in the native code and/or the Java code. Third, errors must be properly handled by both the native methods and the Java code. If an error is generated in the native methods it must be handled properly to prevent the native code from crashing and causing unpredictable problems for the code running on the JVM.

The following sections will detail the implementation of the modular server and client software architecture. The particular advantages of using JNI will be shown and the specific potential difficulties with using native methods will be addressed.

B. SERVER IMPLEMENTATION

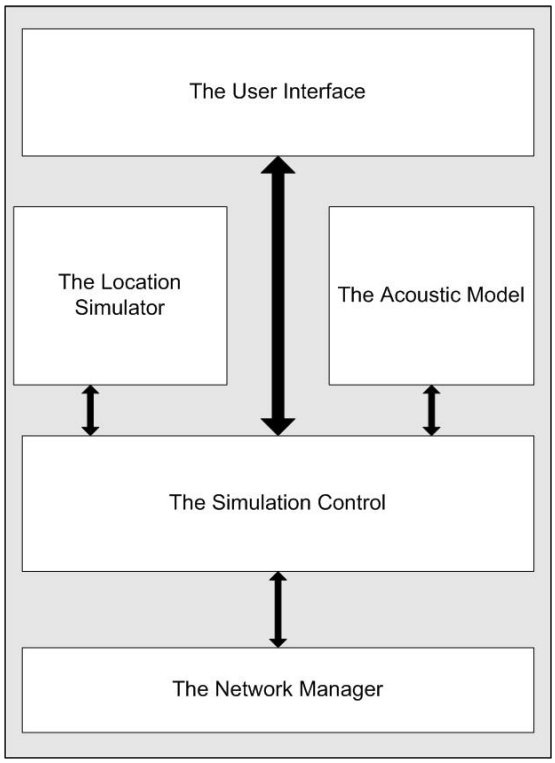

Figure 9: Server Modular Implementation.

As was described in Chapter III and shown above in Figure 9, the software components of the server will be comprised of its core system, the Network Manager, Simulation Control, and User Interface, as well as the Acoustic Simulator and Location Simulator. Together, the core of the system will be light weight and each component will be limited in scope.

The Network Manager provides the network connection management and the User Interface provides a means of displaying system messages and receiving input from the user. The Simulation Control performs simple logic to determine how an incoming/outgoing message should be handled and take the appropriate action. However, the system will still be broken up into logical classes that will enable developers to augment and/or otherwise modify them to suit the user's specific needs with no need to modify large blocks of unrelated code. For example, the developers may wish to modify the User Interface to adapt it to a console, text-based interface vice a GUI. The core of the system is, and should be maintained as, light weight and limited in scope.

These core blocks can be entirely written in Java with little to no performance penalties and maintain the portability of the entire system. The core system will then allow the remaining two components, the Acoustic Simulation and Location Simulation to be plugged in.

These two components can become quite computationally intensive. Also, the desired acoustic model that should be implemented may vary depending on the user's needs. This could require the running of code that was written in C, for example, for performance or legacy reasons. The Acoustic Simulation, due to the requirements discussed in Chapter III and briefly mentioned above, will be considered a separate component from the core system.

The core of the system will allow the developer and user to use almost any type of acoustic model they choose by providing a simple wrapper class. This wrapper class is

essentially a Java interface class, where the developer must implement the given methods. How the developer implements the methods will vary. The developer may use code written in Java. However, the majority of acoustic modeling software is written in C or Fortran. To allow users to make use of these models, the developer only needs to implement the given methods of the interface and also make use of the JNI. Other methods may be written into any class that implements the interface. By providing the skeleton set of required methods, and due to the nature of Object Oriented Programming and JNI, virtually any type of acoustic model may be easily implemented by adapting the wrapper class, through method implementations, to the desired model. The overall amount of code that must be written should be minimal and, since most acoustic modeling software works with the basic data types of float, double, and integer, there is no need to worry about potentially complicated issue of converting between Java objects and other languages' data structures. However, JNI does provide for this if developers desire to make use of it.

The Location Simulator implementation follows the same logic and, therefore, is implemented in a similar manner. The core system will use a Location Simulation interface to allow the developer to easily write or import code into the UAN simulator.

C. CLIENT IMPLEMENTATION

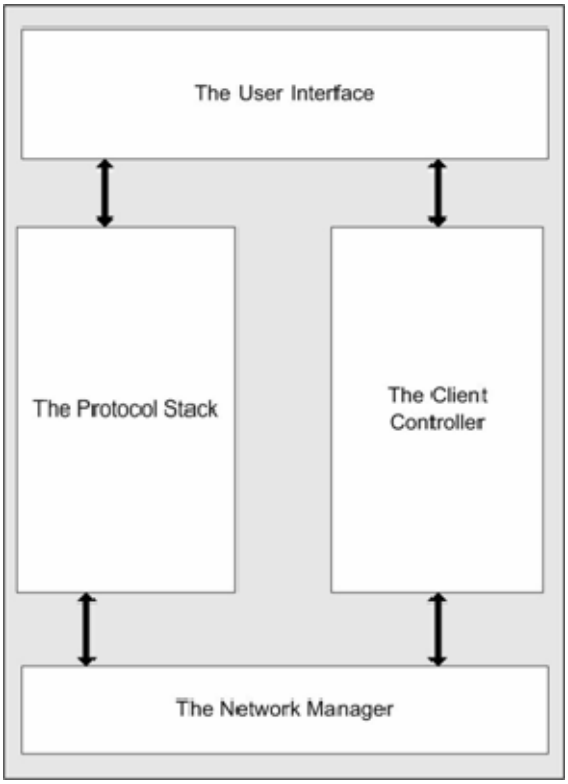

Figure 10: The UAN Client Modular Implementation.

The modular implementation of the client will follow roughly the same principles as the server. The core of the client will be light weight and limited in scope. Figure 10 above diagrams the relationship between the various components.

Those components that will be part of the core client are the Network Manager, User Interface, and Client Controller; their basic functions are similar to their respective counterparts. The component that needs to be discussed further is the Protocol Stack.

The Protocol Stack, as described in Chapter IV, will be used to allow users to test out various protocols in the UAN simulation. The protocols may be written in a variety of

programming languages and, therefore, this component follows the same design as the Acoustic Simulation and Location Simulation components discussed above. The client will make use of a Java interface class that will wrap the actual implementation and make inconsequential to the client the exact details of how it is implemented. To allow the JVM to access the native code, developers will use the JNI to provide the hooks that allow access to the native code.

D. HANDLING THE PROBLEMS

This section will discuss the issues associated with running native code as part of a Java program. Each of the problems discussed above will be handled in turn.

1. Portability

By using native code some may argue that we have now killed one of the most desirable aspects of Java and thereby negated an important claim this and a previous thesis make, namely, that the UAN simulation will be able to run on any hardware platform with a compatible JVM. As recognized earlier, native code restricts the portability to hardware platforms that are similar to the one for which the code was compiled. However, we have split the components into logical units that will maximize portability and performance.

The core components are written entirely in Java. This will provide a consistent GUI or console-based interface across any platform on which this simulation is run and thus improve usability and lower, if not eliminate, any time users will need to become familiar with the interface.

Also, the main simulation logic and network management facilities are maintained across hardware platforms and this eliminates any potential problems that may arise in porting these components. The only portions of the simulation that will require recompilation are the native components of the Acoustic and Location Simulations, and the Protocol Stack. If acoustic models being used are written in a programming language other than Java, then they will require an initial recompilation to ensure the proper JNI headers are incorporated into the source code to allow proper execution when invoked by the JVM. Further, the acoustic model software may be recompiled for the new platform it will be running on to take advantage of compiler optimizations that are available to improve the run-time performance. The acoustic model software would, at most, have to be compiled once for each hardware platform it will run on; the JNI headers and compiler optimizations would be set during this single compilation.

2. Memory Management

Memory management will not present any new problems when incorporating JNI into the simulation. The Java interface class does not require nor expect any type of Java object to be returned; all variables returned are standard data types that require no new memory allocation by the JVM or the native code. The native code should not be calling any Java methods. It is not required by the simulation, nor should any acoustic model expect to have these methods available. Therefore, no memory issues between objects instantiated by the JVM and handed over to the native code should be present. Finally, memory management, when working

with programming languages like C/C++ and Fortran, is always of prime concern and presents no new programming or overall simulation challenges.

3. Error Handling

The last problem that was brought up above was error management. Specifically, the problem arises when an error, fatal or not, develops in the native code. One solution would be to have the native code create and throw an exception back to the JVM. This would help maintain the seamless integration and allow for graceful error handling. However, this method requires some massaging of the native source code to use the proper data structures and methods to allow the native code to create and throw an exception back to the JVM. Another method would be to have the native method return some kind of error code, i.e., a unique integer value to represent a fatal error. This method has two drawbacks: (1) it too would require a rework of the native source code, and (2) it lacks any further detailing explanation as to what the actual error is or what caused it. Yet, this method would be easier to implement since no knowledge of the JNI native methods for throwing exceptions is required.

We present no ideal solution but rather leave it to the developers/users to decide how they would best like to handle errors generated in the native code.

E. THE UAN SIMULATION

Java and the JNI have allowed us to fully implement the framework outlined in the previous two chapters into a

robust, manageable, and, most importantly, into a usable development and evaluation platform for underwater acoustic networking. With a modular server, Figure 9, and client, Figure 10, developers are given much greater power and flexibility when choosing how they would like to evaluate UAN performance. The source code for a small demo server and client implementation are provided in the appendix.

In the following chapter we will validate the framework described in this thesis by implementing an Acoustic Simulation in C and running a simple network ping application as part of the client Protocol Stack.

VI. VALIDATION

A. PROGRAMMING FOR THE SIMULATION

This chapter will detail the software that was developed to demonstrate the ease and practicality of the framework outlined in this thesis. The demonstration software discussed below was developed to use three host systems connected through a network hub utilizing 100Mbs Ethernet links. The first software component and host system that will be shown is the UAN Simulation server, which will be running on a Pentium 4, 2GB RAM, Windows XP (SP2) based desktop computer with Java Virtual Machine version 1.6.0.02. The remaining two hosts will run separate instances of the client software. These two computers are comprised of similar hardware architectures and Java Virtual Machines as the host server, however, these two computers will be running Ubuntu Linux version 7.04.

B. PROGRAMMING THE SERVER

As was discussed in chapter III and V, the server is composed of the following basic software components: User Interface, Network Manager, and Simulation Controller. The server is also composed of dynamically loadable components, the Acoustic and Location Simulators.

1. The Network Manager

The Network Manager that was developed for the demonstration implements a simple, threaded server; the server, once started, will create a new thread and socket

for communicating with each client. Each new communication thread will be maintained by the Network Manager. However, the ability to send and receive communications, simulated UAN acoustic messages, will be handled by the Simulation Controller. When a client connects to the server it will register itself into the simulated environment. This entails setting a node name, location, and course. Node names may be a simple text string, e.g., node1, or a simulated IP Address depending on the users requirements. The node name is used by the simulation server for storing and retrieving information related to a particular client, for example, retrieving the location of simulated node for the location simulator, which will be discussed below. Once the client has been registered with the system, the Simulation Controller is then given control over the communication thread.

2. The Simulation Controller

The Simulation Controller is the nerve center for the UAN Simulation server. All incoming and outgoing simulated UAN acoustic messages as well as user commands will pass through the Simulation Controller. The Simulation Controller implements a set of three first-in-first-out queues: an incoming queue, an outgoing queue, and a user command queue. These queues operate on a round-robin type priority scheme where each queue is examined for messages. If a message exists the message is handled appropriately, then the Simulation Controller moves onto the next queue, etc. If no messages exist in any of the queues, then the

system waits until a new message is received at which point the Simulation Controller is alerted and the appropriate queue is checked.

The incoming queue will handle the simulated acoustic communications that a client sends. Upon receiving the simulated acoustic signal, the Simulation Controller will retrieve the sending nodes location as well as the location of all other nodes in the simulated environment. The sending node acoustic signal and location as well as the location of the each destination node will be sent to the Acoustic Simulator. The Acoustic Simulator will determine the acoustic signal properties, e.g., propagation delay, power, and/or frequency, and then place the derived acoustic signal into the outgoing queue, waiting to be sent to the appropriate simulated node.

The Simulation Controller will also handle user commands. User commands may be simple administrative commands for setting the appropriate network settings, i.e., port number. A user command could also request information or parameters from the Acoustic and Location Simulators. The ability to allow the User Interface to dynamically generate graphical components that will effectively communicate the information and parameters from the Acoustic and Location Simulators is left for future work and is discussed briefly in chapter VII. For this demonstration, the user commands for the server are simple and give the user the ability to set the port number, start/stop the server, and load an Acoustic/Location Simulator.

3. The User Interface

The User Interface allows the developers to design a interface appropriate for the hardware/environment the simulation will be running in. This could be a simple console text based interface or a detailed graphical interface. The interface designed here is a simple GUI that provides a menu of commands for the user to choose from (discussed above) and a scrolling text field. The text field will present informative messages, along with a time stamp, to the user. Figure 11 shows the server GUI.

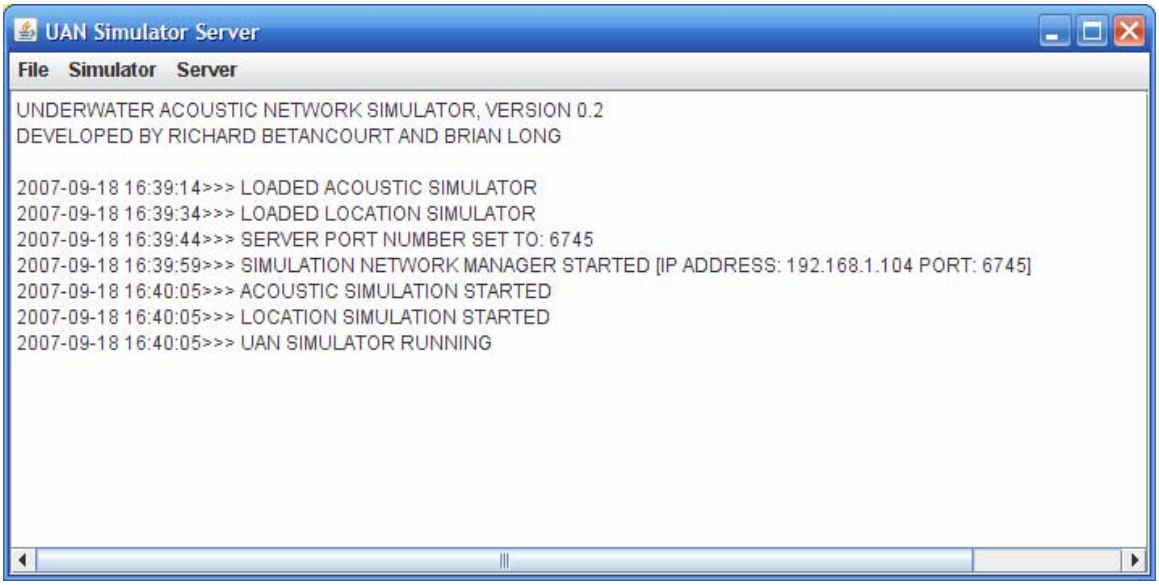

Figure 11: The UAN Simulation Server GUI.

4. The Acoustic and Location Simulators

The Acoustic and Location Simulators are the dynamically loadable components of the UAN simulation and could possibly be written in another programming language besides Java. For the purposes of this thesis, a spherical model of sound spreading with an average sound speed of 1500

m/s was developed in C. The choice of this simple model was chosen to highlight the ease of incorporating the Java Native Interface into the simulation. The Location Simulator is written in Java.

To use the C program of the spherical model requires the generation of system library; on Microsoft Windows this is a Dynamically Linked Library (DLL) file and in Unix type operating systems this is a “lib*.so” file. In this demonstration the server is running on a Microsoft Windows based computer and therefore requires generating a DLL. Regardless of the type of library file, the process will require four steps: (1) creating a Java wrapper class that will load and call the native C program with the appropriate parameters, (2) generating the appropriate C header files, (3) compiling the Java wrapper source code to generate a “.class” file, and (4) compiling the library file. Once these steps are completed, the developers place the generated library file into the appropriate directory. For Microsoft Windows, this is the “\Windows\system32\” directory.

Built into the Simulation Controller is the ability to dynamically load Acoustic and Location Simulators. Users can choose which acoustic simulator they wish to employ in the UAN simulation by selecting the appropriate “.class” file, the wrapper class that was generated in step (3), that will call the appropriate library file. Figure 12 shows an example of a user selecting the “SphericalAcousticModel.class” file, which simulates the model we want to use for this demonstration.

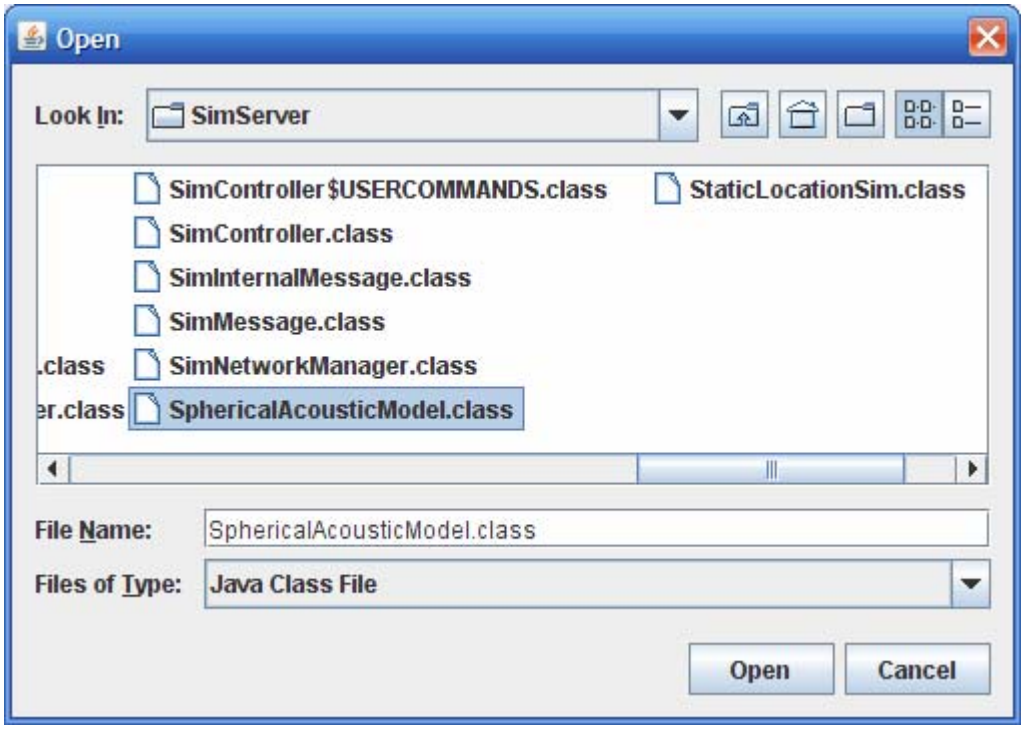

Figure 12: Loading an Acoustic Simulation.

The process of developing and loading a Location Simulator is the same.

C. PROGRAMMING THE CLIENT

The UAN simulation client employs similar elements to the UAN simulation server. The components for the client include the Network Manager, Simulation Client Controller, the User Interface, and the Protocol Stack. The Protocol Stack is where developers have the ability to implement desired protocol algorithms to suit there requirements and dynamically load into the simulation. The other components of the client will be briefly described.

1. User Interface and Network Manager

The User Interface for the client is similar to the server. A menu of commands is given to the user and a scrollable text field is also provided for informative simulation messages.

The Network Manager that is implemented will allow users to dynamically choose the server port and IP address and then connect to the server. Once a socket connection is established with the UAN simulation server, the client node name, location, and course are sent to the server to be registered. Once registered, the client node has established a connection, control over the socket is given to the Simulation Client Controller, and the client node is then ready to begin simulated acoustic communications. Figure 13, shows the results of a client node connecting to a server and figure 14 shows the server side results.

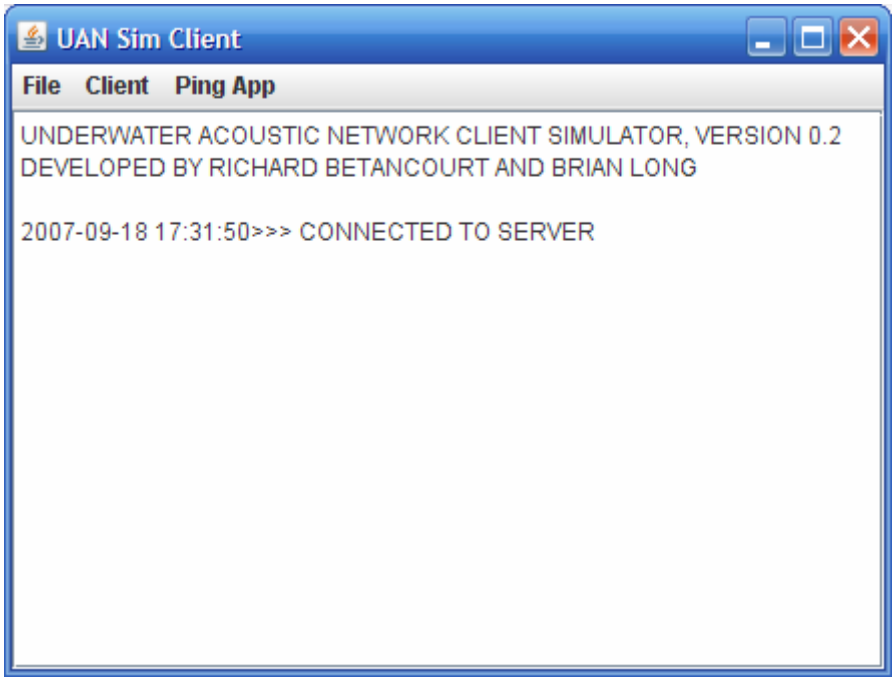

Figure 13: A Client Connecting to the Server.

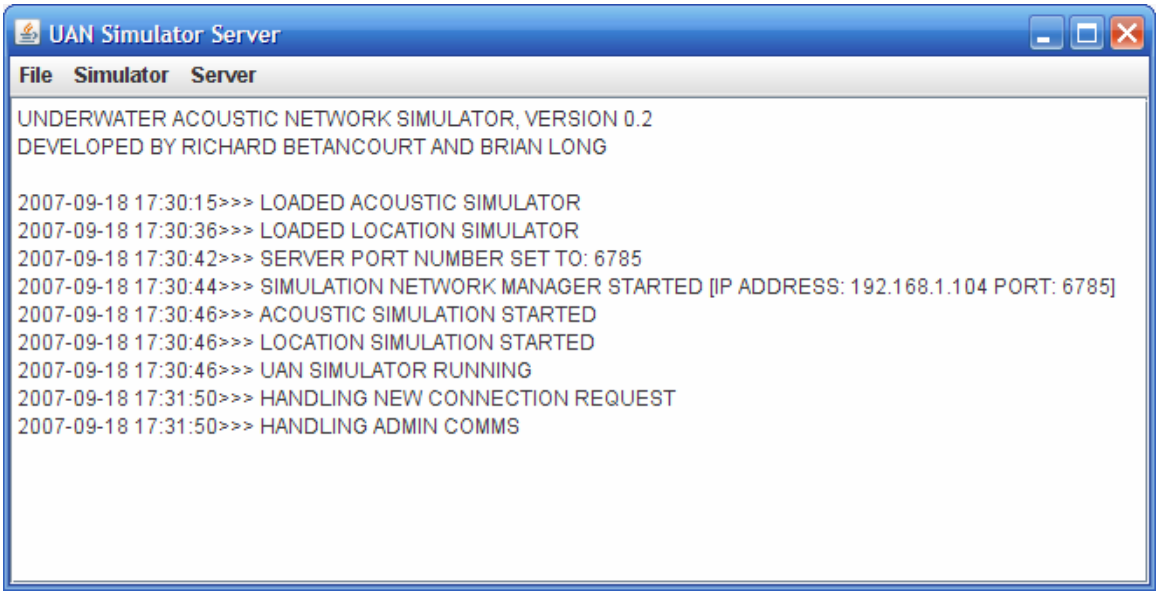

Figure 14: The Server Response to Client Connection.

2. The Simulation Client Controller

The Simulation Client Controller performs similar functions that the server Simulation Controller performs. A pare of FIFO queues are used to handle incoming and outgoing communications. Incoming communications are split into two main categories: administrative and simulation or acoustic signal messages.

The administrative messages will be handled by the Simulation Client Controller. These messages include those needed to register the client on the server and disconnection requests and responses to and from the server.

Simulation acoustic signal messages require the client to simulate the required propagation delay. The incoming acoustic signal will be placed in its own thread and will remain dormant until required amount of time, the calculated propagation delay as determined by the Acoustic Simulator minus the network delay, has expired. The acoustic signal

will then be passed into the Protocol Stack. The process of determining if an acoustic signal collision occurs is handled by the Protocol Stack.

3. The Protocol Stack

The Protocol Stack allows developers to implement desired network protocol algorithms and dynamically load them. The network protocol algorithms may also be written in another language other than Java and, therefore, may require the use of the Java Native Interface. Developing and implementing the Protocol Stack is carried out in the same manner as the Acoustic and Location Simulators of the UAN server. Once the Protocol Stack has been developed, it is loaded and then can be used by the client. The demonstration Protocol Stack implements a very simple “ping” application. A client node selects which node to send a ping to and then transmits the simulated ping. Figures 15 and 16 show a very simple “ping” application between two simulated UAN client nodes.

Figure 15: Node2 Sending Ping to Node1.

Figure 16: Node1 Receiving Ping from Node2.

This chapter showed the feasibility of implementing the framework described and utilizing the Java Native Interface API. In the following chapter we will conclude the discussion of the framework and point the way towards future development and evaluation.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSION AND FUTURE WORK

A. CONCLUSION

Building on the original distributed UAN simulation developed by Brian Long and using the advanced Java JNI API, a new framework has been developed to provide an ability to incorporate pre-built software. The ability to use these pre-built software components will greatly increase productivity by decreasing the amount of software development time and allow users to concentrate on obtaining and evaluating results for the desired underwater acoustic environment.

As discussed, the more accurate underwater acoustic models as developed by the scientists in the physics area are not being used by network developers. Most of these models are computationally intensive and require a greater amount of the host system's resources. Instead of these models, UAN developers are using modified terrestrial radio frequency models, which brings into question the fidelity of results obtained. Also, the ocean environment is not modeled, which is a key factor in UAN performance. The framework presented in this thesis allows UAN developers to choose the acoustic model with the desired level of accuracy and incorporate it into the simulation with little or no need to alter the model's source code and obtain potentially more accurate results

The other area the framework developed here addresses its ability to support various protocol stacks. The protocols, written in various languages, can be used with

the UAN simulation with little or no required alteration of the original source code. As is the case with some other network simulation tools, the protocols that will be used in the simulation require them to be written in another language or script. Or, preexisting protocols may exist which developers desire to implement and evaluate in the UAN environment. The framework developed here allows the use of protocols written in the language of the developer's choice.

B. FUTURE WORK

An important area that needs to be evaluated is the means of coordinating communications timing between the clients and server. If the simulation server and clients are all running on the same host, the problem of timing is so small it can be safely ignored. And even in a small LAN, where the simulation is not competing for bandwidth with other non-simulation users, the physical network delay may still be considered relatively small and may not need to be taken into account. However, the distributed nature of this simulation is an important element and the possibility of users stretching across campuses, where the network resources must be shared with other non-simulation users, is more likely, the network delay must be taken into account as well as the variability or skew of host system clocks. Handling the latter problem will require the synchronization of the client clocks. This will compensate for host clock skew when timing (simulated propagation delay) of sent simulation messages is handled by the clients. An obvious means of synchronization would be to use a time server. This server could be incorporated into the simulation or, as is the case on most large networks, use a preexisting time

server. Another method that would take into account the network delay and not require time synchronization would be to implement a type of echo protocol. This echo protocol would periodically send an echo ping to all attached clients. The round trip time for each client would be determined and a running average calculated. Individual clients would have an average network delay value associated with itself and the serve. This value would then be applied to the delay time that the acoustic model determines and sends to the destination client(s). For example, assume the average network delay for client A is d1 milliseconds and client B is d2 milliseconds. The server will take into account the average network delay between nodes when determining the propagation delay, if X is the propagation delay, in milliseconds, then X – (d1 + d2) would be the simulated propagation delay that B would need to wait, using its own clock as a reference, before sending the message to the simulated protocol stack.

Another area that needs to be looked at is the ability to allow users to dynamically alter key simulation parameters. Developers of the acoustic models, location simulations, and clients should develop an XML specification for the controls that are available in the models, the parameters that are modifiable, and allow the clients and servers to populate a user interface, GUI, with the appropriate controls. For example, an acoustic model may take as a parameter the variability of the ambient noise level or alter the depths and speeds of the sound speed profile. Another example would be to allow the client users to alter the power and frequency of the acoustic model or other protocol stack parameters. To accomplish this, an XML

document should be developed that would allow the developers to specify what the parameters are (i.e., name of the parameter and type), minimum and maximum values, etc. With a standardized XML document used for parameter descriptions, a parser could be added to the server and clients that would allow them, upon loading of the specific acoustic model/protocol stack, to generate the appropriate GUI controls. The ability to generate these controls would greatly increase the user's ability to dynamically alter environmental and client settings and more accurately model the variability found in the ocean.

Performance of the proposed UAN simulation framework needs to be measured. To accomplish this, acoustic physics models and protocols need to be incorporated into the framework and their performance evaluated. The evaluation of performance needs to be broken up according to various degrees of model complexity. This will generate key performance metrics and allow developers to focus on specific areas, i.e., simulation protocol overhead, and generate required changes to the system. The application of grid computing as a distributed system of the acoustic model should also be examined as a means of improving the performance of these computationally complex models.

Overall, a distributed and modular UAN simulation framework has been proposed. This framework needs to be used, metrics obtained, and from there future improvements of this framework may be incorporated and provide those exploring the ocean depths with a robust simulation tool for deploying underwater networks.

APPENDIX

A. SIMULATION SERVER SOURCE CODE

1. SimController.java

```
package SimServer;

import Utilities.ICommandUI;
import Utilities.ISimController;
import Utilities.SimCommsThread;
import Utilities.SimInternalMessage;
import Utilities.Location;
import Utilities.LocationException;
import java.util.Enumeration;
import java.util.NoSuchElementException;
import java.util.Hashtable;
import Utilities.AcousticSignal;
import Utilities.SimMessage;
import Utilities.UANSimException;
import java.util.LinkedList;
import java.util.Random;
import java.io.IOException;
import java.lang.Math;

public class SimController implements
Utilities.ISimController
{
    public static enum USERCOMMANDS
    {
        /**Start the simulation*/
        STARTSIM,

        /**Stop the simulation*/
        STOPSIM,

        /**Start the network manager*/
        STARTSERVER,

        /**Stop the network manager*/
        STOPSERVER,

        /**Request server information*/
```

```
        SERVERINFO,

        /**Set the server port number*/
        SETSERVERPORT,

        /**Request all client information*/
        ALLCLIENTINFO


    }

    private static final int MAXRAND =
(int)Math.pow(2.0,31.0);

    private IAcousticSim as;

    private ICommandUI cui;

    private ILocationSim ls;

    private SimNetworkManager snm;

    private Hashtable<String,SimCommsThread> clients;

    private LinkedList<SimInternalMessage> comms_inputqueue;

    private LinkedList<SimInternalMessage>
action_inputqueue;

    private LinkedList<SimInternalMessage> admin_inputqueue;

    private LinkedList<SimInternalMessage> outputqueue;

    private InputQueueHandler iqh;

    private OutputQueueHandler oqh;

    private Random rand;

    /** Creates a new instance of SimController */
    public SimController(ICommandUI ui)
    {
        this.cui = ui;

        this.as = null;
        this.ls = null;
```

```
        this.clients = new Hashtable<String,
SimCommsThread>();
        this.action_inputqueue = new
LinkedList<SimInternalMessage>();
        this.admin_inputqueue = new
LinkedList<SimInternalMessage>();
        this.comms_inputqueue = new
LinkedList<SimInternalMessage>();
        this.outputqueue = new
LinkedList<SimInternalMessage>();

        this.iqh = new
InputQueueHandler(this.comms_inputqueue,
this.admin_inputqueue, this.action_inputqueue, this);
        this.oqh = new OutputQueueHandler(this,
this.outputqueue);

        this.snm = new SimNetworkManager();
        this.snm.setController(this);

        this.rand = new Random();
    }

    public void setCUI(ICommandUI ui)
    {
        this.cui = ui;
    }

    public void setAcousticSimulator(IAcousticSim as)
    {
        this.as = as;
        this.iqh.setAcousticSim(this.as);
    }

    public void setLocationSimulator(ILocationSim ls)
    {
        this.ls = ls;
        this.iqh.setLocationSim(this.ls);
    }



    /**
     * Start the UAN simulator.
     */
```

```
    public void startSimulator()
    {
        /**
         * Before we try and start anything, make sure we
have an acoustic and
         * location simulators loaded.
         */
        if(this.as == null)
        {
            String err_msg = "ERROR: UNABLE TO START
SIMULATOR - NO ACOUSTIC SIMULATOR LOADED.\n";
            err_msg += "PLEASE LOAD AN ACOUSTIC SIMULATOR.";
            this.cui.errorMessageHandler(err_msg);
            return;
        }
        if(this.ls == null)
        {
            String err_msg = "ERROR: UNABLE TO START
SIMULATOR - NO LOCATION SIMULATOR LOADED.\n";
            err_msg += "PLEASE LOAD A LOCATION SIMULATOR.";
            this.cui.errorMessageHandler(err_msg);
            return;
        }

        /**Start the network manager and inform the user*/
        if(!this.snm.isRunning())
        {
            this.snm.startNetworkManager();
            String snmStatus = "SIMULATION NETWORK MANAGER
STARTED [IP ADDRESS: ";
            snmStatus += this.snm.getIPAddress() + " PORT: "
+ this.snm.getPort() + "]";
            this.cui.generalMessageHandler(snmStatus);
        }

        /**Start the input and output queue handlers*/
        this.iqh.start();
        this.oqh.start();

        /**Start the acoustic and location simulators and
inform the user*/
        this.as.startAcousticSim();
        this.cui.generalMessageHandler("ACOUSTIC SIMULATION
STARTED");

        this.ls.startLocationSim();
```

```
        this.cui.generalMessageHandler("LOCATION SIMULATION
STARTED");

        /**Everythins has started, inform the user*/
        this.cui.generalMessageHandler("UAN SIMULATOR
RUNNING");


    }

    public void stopSimulator()
    {
        //TODO: Add code to send admin messages to all
connected clients to inform
        //      them that the server is shutting down.

        /**Stop simulation network manager*/
        if(this.snm.isRunning())
        {
            this.snm.stopNetworkManager();
            this.cui.generalMessageHandler("SIMULATION
NETWORK MANGER STOPPED");
        }
        else
        {
            //this.cui.generalMessageHandler("SIMULATION WAS
NOT RUNNING");
            return;
        }

        /**Lock all the queues and empty them*/
        synchronized(this.action_inputqueue)
        {
            this.action_inputqueue.clear();
        }

        synchronized(this.admin_inputqueue)
        {
            this.admin_inputqueue.clear();
        }

        synchronized(this.comms_inputqueue)
        {
            this.comms_inputqueue.clear();
        }
```

```
        /**Stop the input and output queue handlers*/
        this.iqh.setRunning(false);
        this.oqh.setRunning(false);



        /**Stop all the communication threads*/
        Enumeration sct_enum = this.clients.elements();
        while(sct_enum.hasMoreElements())
        {
            try
            {
                SimCommsThread temp =
(SimCommsThread)sct_enum.nextElement();
                temp.shutdownSocket();
            }
            catch(NoSuchElementException nsee)
            {

            }
        }

        /**Clear the hash table of all keys*/
        this.clients.clear();

        /**Stop the acoustic and location simulations*/
        this.as.stopAcousticSim();
        this.ls.stopLocationSim();

        this.cui.generalMessageHandler("ACOUSTIC AND
LOCATION SIMULATION STOPPED");


    }

    public void processInComms(SimInternalMessage sim)
    {
        /**Add the incoming message to the comms queue*/
        synchronized(this.comms_inputqueue)
        {
            this.comms_inputqueue.add(sim);
            this.iqh.interrupt();
        }
    }

    public void processOutComms(SimInternalMessage sim)
```

```
    {
        synchronized(this.outputqueue)
        {
            this.outputqueue.add(sim);
            this.oqh.interrupt();
        }
    }

    public void processAction(SimInternalMessage sim)
    {
        /**Add the incoming message to the action queue*/
        synchronized(this.action_inputqueue)
        {
            this.action_inputqueue.add(sim);
            this.iqh.interrupt();
        }
    }

    public void processAdmin(SimInternalMessage sim)
    {
        /**Add the incoming message to the admin queue*/
        synchronized(this.admin_inputqueue)
        {
            this.admin_inputqueue.add(sim);
            this.iqh.interrupt();
        }
    }

    public String processUserRequest(String request,
String... choices)
    {
        //TODO: Add code to send input from the server users
through the
        //      ICommandUI interface.
        return null;
    }

    public void processUserInput(USERCOMMANDS uc, Object ...
input)
    {
        //TODO: Add code to handle all the possible inputs
we could get.

        switch(uc)
        {
            case STARTSIM:
```

```
                this.startSimulator();
                break;
            case STOPSIM:
                this.stopSimulator();
                break;
            case STARTSERVER:
                this.startNetworkManager();
                break;
            case STOPSERVER:
                this.stopNetworkManager();
                break;
            case SETSERVERPORT:
                try
                {
                    int portnum = (Integer)input[0];
                    if(portnum <= 0 || portnum >= 65536)
                    {
                        this.cui.errorMessageHandler("ERROR:
"+portnum+" IS NOT A VALID PORT NUMBER.");
                    }
                    else
                    {
                        try
                        {
                            if(this.snm.isRunning())
                            {

this.snm.stopNetworkManager();

                                this.snm = new
SimNetworkManager();

this.snm.setController(this);
                                this.snm.setPort(portnum);

this.cui.generalMessageHandler("SERVER PORT NUMBER SET TO:
"+portnum);

this.snm.startNetworkManager();
                            }
                            else
                            {
                                this.snm.setPort(portnum);

this.cui.generalMessageHandler("SERVER PORT NUMBER SET TO:
"+portnum);
```

```
                        }
                    }
                    catch(IOException ex)
                    {

this.cui.errorMessageHandler("ERROR: IOException when
setting port number.");
                    }

                }
                break;
            }
            catch(NumberFormatException nfe)
            {
                this.cui.errorMessageHandler("ERROR:
"+input[0]+" IS NOT A NUMBER.");
                break;
            }
        case SERVERINFO:
            String status = "The server is ";
            boolean serverrunning =
this.snm.isRunning();
            if(serverrunning)
            {
                status += "running.\n";
            }
            else
            {
                status += "not running.\n";
            }

            status += "The server IP address is: " +
this.snm.getIPAddress() + "\n";
            status += "The server port number is: " +
Integer.toString(this.snm.getPort());

            this.cui.generalMessageHandler(status);
            break;
        default:
            this.cui.errorMessageHandler("ERROR: UNKNOWN
COMMAND.");
            break;
    }
}

public void processFatalError(String msg)
```

```
    {
        this.cui.errorMessageHandler(msg);
    }

    public void addNewClient(SimCommsThread sct)
    {
        /**Randomly generate a name for this client*/
        String nodeName = "Node:";
        do
        {
            nodeName +=
Integer.toString(rand.nextInt(MAXRAND));
            if(this.clients.containsKey(nodeName))
                continue;
            break;
        }while(true);

        /**Assign the name to the simulation comms thread*/
        sct.setName(nodeName);
        //sct.start();

        /**Add it to the clients hashtable and start the
thread*/
        this.clients.put(nodeName,sct);

        this.cui.generalMessageHandler("HANDLING NEW
CONNECTION REQUEST");
    }

    /**
     * Returns a reference to the simulation communications
thread that is
     * connected to the client with the given node name.
     * @param name The name of the node we want the
communications thread for.
     * @return The communicaiton thread for the given node.
     */
    public SimCommsThread getCommsThread(String name)
    {
        return this.clients.get(name);
    }

    /**
     * This method will handle all administrative messages
coming from clients.
     */
```

```
    public void adminHandler(String name, String sim)
    {
        this.cui.generalMessageHandler("HANDLING ADMIN
COMMS");
        String[] input;
        input = sim.split(" ");
        if(input[0].equals("CLIENT_NAME"))
        {


            SimCommsThread sct1 = this.clients.get(name);
            sct1.setName(input[1]);
            this.clients.remove(name);
            this.clients.put(input[1],sct1);

            SimInternalMessage ms = new
SimInternalMessage();
            ms.setMessageType(SimMessage.MESSAGETYPE.ADMIN);
            ms.setContent("NAME_OK");
            ms.setNodeName(input[1]);
            synchronized(this.outputqueue)
            {
                this.outputqueue.add(ms);
                this.outputqueue.notify();
            }

        }
        else if(input[0].equals("SET_LOCATION"))
        {
            Location loc = new Location();
            try
            {

loc.setLatitude(Double.parseDouble(input[1]));

loc.setLongitude(Double.parseDouble(input[2]));
                loc.setDepth(Integer.parseInt(input[3]));
            }
            catch(LocationException le)
            {
                //
            }

            this.ls.processLocation(name, loc);
```

```
            SimInternalMessage ms = new
SimInternalMessage();
            ms.setMessageType(SimMessage.MESSAGETYPE.ADMIN);
            ms.setContent("LOCATION_OK");
            ms.setNodeName(name);
            synchronized(this.outputqueue)
            {
                this.outputqueue.add(ms);
                this.outputqueue.notify();
            }



        }
        else if(input[0].equals("DISCONNECT"))
        {
            SimCommsThread sct1 = this.clients.get(name);
            SimMessage sm = new SimMessage();
            sm.setMessageType(SimMessage.MESSAGETYPE.ADMIN);
            sm.setContent("DISCONNECT_OK");

            sct1.sendMessage(sm);
            sct1.shutdownSocket();

            this.clients.remove(name);
            this.ls.removeLocation(name);

        }
    }

    /**
     * The method will ensuring that data, usually an
acoustic signal coming back
     * from the acoustic simulator plug-in (IAcousticSim),
will be processed, i.e.
     * queued up for transmition to the client node.
     * @param sim The simulation internal message that
contains the data to send
     *            to the node encapsulated in the
sim.nodeName data member.
     */
    public void processDataToNode(SimInternalMessage sim)
    {
        synchronized(this.outputqueue)
        {
            this.outputqueue.add(sim);
```

```
        }
    }

    private void startNetworkManager()
    {
        this.snm.startNetworkManager();
        String snmStatus = "SIMULATION NETWORK MANAGER
STARTED [IP ADDRESS: ";
        snmStatus += this.snm.getIPAddress() + " PORT: " +
this.snm.getPort() + "]";
        this.cui.generalMessageHandler(snmStatus);
    }

    private void stopNetworkManager()
    {
        this.snm.stopNetworkManager();
        this.cui.generalMessageHandler("SIMULATION NETWORK
MANGER STOPPED");
    }

    public Enumeration<String> getAllClients()
    {
        return this.clients.keys();
    }

    public void processError(String msg)
    {
        this.cui.errorMessageHandler(msg);
    }

}
```

2. SimNetworkManager.java

```
package SimServer;

import Utilities.*;
import Utilities.ISimNetworkManager;
import Utilities.SimCommsThread;
import Utilities.ISimController;
import java.util.*;
import java.net.*;
import java.io.*;

/**
 *
```

```
 * @author richard betancourt
 */
public class SimNetworkManager extends Thread implements
Utilities.ISimNetworkManager
{

    // Private Data Members

    /**Max number of nodes*/
    private int maxnodes = 10;

    /**Server port number*/
    private int portnumber = 2875;

    /**The server socket that will listen for nodes*/
    private ServerSocket serverSocket;

    /**Status of the server*/
    private boolean serverRunning;

    /**Flag to keep the server running*/
    private boolean keepServerRunning;

    /**
     * The simulator controller object to which all newly
maid connection threads will
     * be passed to.
     */
    private SimController controller;


    /** Default constructor. Creates a new instance of
SimNetworkManager */
    public SimNetworkManager()
    {
        this.controller = null;
        this.serverSocket = null;
        this.serverRunning = false;

    }

    /**
     * Set the port number.
     * @param port_num The port number on which the server
will listen for new connections from clients.
     */
```

```
    public void setPort(int port_num) throws IOException
    {
        if(port_num != this.portnumber)
        {
            this.portnumber = port_num;
        }

    }


    /**
     * Set the Controller object that this NetworkManager
will communicate with.
     * @param c The simulation controller to which this
network manager will pass
     *          newly created threads for the controller to
use for communications.
     */
    public void setController(SimController c)
    {
        this.controller = c;
    }


    /**
     * Get the server port number.
     * @return The portnumber on which this server is
listening.
     */
    public int getPort()
    {
        return this.portnumber;
    }

    /**
     * Get the IP address of the this server.
     * @return The IP address that this client is on.
     */
    public String getIPAddress()
    {
        String ipaddress =
this.serverSocket.getInetAddress().toString();
        int index = ipaddress.indexOf("/");
        if(index == -1)
            return ipaddress;
        else
```

```
            return ipaddress.substring(index+1);
    }

    /**
     *Start the server.
     */
    public void startNetworkManager()
    {

        try
        {
            this.serverSocket = new
ServerSocket(this.portnumber,50,InetAddress.getLocalHost());
            this.keepServerRunning = true;
            this.start();

        }
        catch(IOException e)
        {
            System.out.println("IOException in
StartServer()");
            e.printStackTrace();

        }

    }

    public boolean isRunning()
    {
        return this.serverRunning;
    }

    /**
     * Stop the server.
     */
    public void stopNetworkManager()
    {
        this.keepServerRunning = false;
    }

    /**
     * The main method of this class which will loop
continuously waiting for
     * client connections.  A socket timeout is set on which
the server will get
     * interrupted and then check if it should keep running.
```

```
     */
    public void run()
    {
        this.serverRunning = true;
        try
        {
            this.serverSocket.setSoTimeout(500);
        }
        catch(SocketException se)
        {
            //TODO: Add code here to handle the socket
exception
        }

        while(this.keepServerRunning)
        {
            try
            {
                Socket new_socket =
this.serverSocket.accept();
                new_socket.setKeepAlive(true);
                SimCommsThread new_thread = new
SimCommsThread(new_socket, "", this.controller);
                new_thread.start();
                this.controller.addNewClient(new_thread);
            }
            catch(SocketTimeoutException ste)
            {
                continue;
            }
            catch(IOException e)
            {
                this.serverRunning = false;
                String err_msg = "IOException in run() of
SimNetworkManager\n";
                err_msg += e.getMessage() + "\n";
                err_msg += "Server stopped.";

                this.controller.processFatalError(err_msg);

            }

        }

        try
        {
```

```
            this.serverSocket.close();
        }
        catch(IOException ex)
        {
            //Need to do something here.
        }
        this.serverRunning = false;

    }

    public void setController(ISimController controller)
    {
        this.controller = (SimController)controller;
    }
}
```

3. JFrameCommandGUI.java

```
package SimServer;

import Utilities.*;
import Utilities.ICommandUI;
import java.io.File;
import java.net.*;
import java.util.Calendar;
import java.util.Date;
import javax.swing.JOptionPane;
import java.text.SimpleDateFormat;

/**
 *
 * @author  richard betancourt
 */
public class JFrameCommandGUI extends javax.swing.JFrame
implements ICommandUI
{
    private static final String GREETING = "UNDERWATER
ACOUSTIC NETWORK SIMULATOR, VERSION 0.2\nDEVELOPED BY
RICHARD BETANCOURT AND BRIAN LONG\n\n";

    private static final String DATE_FORMAT = "yyyy-MM-dd
HH:mm:ss";

    private static SimpleDateFormat sdf;

    /**The main simulation controller*/
```

```
    private SimController controller;

    /**A string which will be used to hold information
messages that are coming
     * back from the simulation controller and given to the
jEditorPane to display
     * for the users.
     */
    private String sim_info;




    /** Creates new form JFrameCommandGUI */
    public JFrameCommandGUI()
    {
        initComponents();

        /**Initialized the simulation controller*/
        this.controller = new SimController(this);

        sim_info = this.GREETING;

        this.jEditorPane_InfoPane.setText(sim_info);

        this.sdf = new SimpleDateFormat(DATE_FORMAT);


    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="
Generated Code ">
    private void initComponents()
    {
        jScrollPane1 = new javax.swing.JScrollPane();
        jEditorPane_InfoPane = new
javax.swing.JEditorPane();
        jMenuBar1 = new javax.swing.JMenuBar();
        jFileMenu = new javax.swing.JMenu();
        jMenuItem_Exit = new javax.swing.JMenuItem();
```

```
        jSimulatorMenu = new javax.swing.JMenu();
        jMenuItem_Start = new javax.swing.JMenuItem();
        jMenuItem_Stop = new javax.swing.JMenuItem();
        jMenuItem_LoadAcousticSim = new
javax.swing.JMenuItem();
        jMenuItem_LoadLocationSim = new
javax.swing.JMenuItem();
        jServerMenu = new javax.swing.JMenu();
        jMenuItem_StartNetMan = new javax.swing.JMenuItem();
        jMenuItem_StopNetMan = new javax.swing.JMenuItem();
        jMenuItem_GetServerInfo = new
javax.swing.JMenuItem();
        jMenuItem_SetPortNum = new javax.swing.JMenuItem();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON
_CLOSE);
        setTitle("UAN Simulator Server");
        addWindowListener(new java.awt.event.WindowAdapter()
        {
            public void
windowClosing(java.awt.event.WindowEvent evt)
            {
                formWindowClosing(evt);
            }
        });

        jEditorPane_InfoPane.setEditable(false);
        jScrollPane1.setViewportView(jEditorPane_InfoPane);

        jFileMenu.setText("File");
        jMenuItem_Exit.setText("Exit");
        jMenuItem_Exit.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_ExitActionPerformed(evt);
            }
        });

        jFileMenu.add(jMenuItem_Exit);

        jMenuBar1.add(jFileMenu);
```

```
        jSimulatorMenu.setText("Simulator");
        jMenuItem_Start.setText("Start");
        jMenuItem_Start.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_StartActionPerformed(evt);
            }
        });

        jSimulatorMenu.add(jMenuItem_Start);

        jMenuItem_Stop.setText("Stop");
        jMenuItem_Stop.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_StopActionPerformed(evt);
            }
        });

        jSimulatorMenu.add(jMenuItem_Stop);

        jMenuItem_LoadAcousticSim.setText("Load Acoustic
Sim");
        jMenuItem_LoadAcousticSim.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {

jMenuItem_LoadAcousticSimActionPerformed(evt);
            }
        });

        jSimulatorMenu.add(jMenuItem_LoadAcousticSim);

        jMenuItem_LoadLocationSim.setText("Load Location
Sim");
        jMenuItem_LoadLocationSim.addActionListener(new
java.awt.event.ActionListener()
```

```
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {

jMenuItem_LoadLocationSimActionPerformed(evt);
            }
        });

        jSimulatorMenu.add(jMenuItem_LoadLocationSim);

        jMenuBar1.add(jSimulatorMenu);

        jServerMenu.setText("Server");
        jMenuItem_StartNetMan.setText("Start Network
Server");
        jMenuItem_StartNetMan.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_StartNetManActionPerformed(evt);
            }
        });

        jServerMenu.add(jMenuItem_StartNetMan);

        jMenuItem_StopNetMan.setText("Stop Network Server");
        jMenuItem_StopNetMan.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_StopNetManActionPerformed(evt);
            }
        });

        jServerMenu.add(jMenuItem_StopNetMan);

        jMenuItem_GetServerInfo.setText("Get Server Info");
        jMenuItem_GetServerInfo.addActionListener(new
java.awt.event.ActionListener()
        {
```

```
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_GetServerInfoActionPerformed(evt);
            }
        });

        jServerMenu.add(jMenuItem_GetServerInfo);

        jMenuItem_SetPortNum.setText("Set Port Number");
        jMenuItem_SetPortNum.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_SetPortNumActionPerformed(evt);
            }
        });

        jServerMenu.add(jMenuItem_SetPortNum);

        jMenuBar1.add(jServerMenu);

        setJMenuBar(jMenuBar1);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
            .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 400, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
            .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 279, Short.MAX_VALUE)
        );
        pack();
    }// </editor-fold>
```

```
    private void
jMenuItem_LoadLocationSimActionPerformed(java.awt.event.Acti
onEvent evt)
    {
        ILocationSim ias;
        URL[] fileURL = new URL[1];
        File locationclass;

        /**Display a file chooser dialog and load the class
file*/
        Loader loader = new
Loader(Loader.FILETYPES.JAVA_CLASS);
        locationclass = loader.openClass();
        if(locationclass == null)
            return;
        try
        {
            fileURL[0] =  locationclass.toURI().toURL();
        }
        catch(MalformedURLException murle)
        {
            this.errorMessageHandler("ERROR: MALFORMED URL
WHEN TRYING TO LOAD FILE.");
            return;
        }

        /**Dynamically load the class and pass it to the
simulation controller*/
        URLClassLoader urlloader = new
URLClassLoader(fileURL);
        int index =
locationclass.getName().indexOf(".class");
        try
        {
            Class locationsim =
urlloader.loadClass("SimServer." +
locationclass.getName().substring(0,index));
            if(locationsim != null)
            {
                ias =
(ILocationSim)locationsim.newInstance();
                this.controller.setLocationSimulator(ias);
                this.generalMessageHandler("LOADED LOCATION
SIMULATOR");
            }
            else
```

```
            {
                this.generalMessageHandler("FAILED TO LOAD
LOCATION SIMULATOR");
            }
        }
        catch(ClassNotFoundException cnfe)
        {
            this.generalMessageHandler("ERROR: CLASS NOT
FOUND EXCEPTION WHEN TRYING TO LOAD LOCATION SIM.");
        }
        catch(InstantiationException ie)
        {
            this.generalMessageHandler("ERROR: INSTANTIATION
EXCEPTION WHEN LOADING LOCATION SIM.");
        }
        catch(IllegalAccessException iae)
        {
            this.generalMessageHandler("ERROR: ILLEGAL
ACCESS EXCEPTION WHEN LOADING LOCATION SIM.");
        }
    }

    private void
jMenuItem_LoadAcousticSimActionPerformed(java.awt.event.Acti
onEvent evt)
    {
        IAcousticSim ias;
        URL[] fileURL = new URL[1];
        File acousticclass;

        /**Display a file chooser dialog and load the class
file*/
        Loader loader = new
Loader(Loader.FILETYPES.JAVA_CLASS);
        acousticclass = loader.openClass();
        if(acousticclass == null)
            return;
        try
        {
            fileURL[0] =  acousticclass.toURI().toURL();
        }
        catch(MalformedURLException murle)
        {
            this.errorMessageHandler("ERROR: MALFORMED URL
WHEN TRYING TO LOAD FILE.");
            return;
```

```
        }

        /**Dynamically load the class and pass it to the
simulation controller*/
        URLClassLoader urlloader = new
URLClassLoader(fileURL);
        int index =
acousticclass.getName().indexOf(".class");
        try
        {
            Class acousticsim =
urlloader.loadClass("SimServer." +
acousticclass.getName().substring(0,index));
            if(acousticsim != null)
            {
                ias =
(IAcousticSim)acousticsim.newInstance();
                this.controller.setAcousticSimulator(ias);
                this.generalMessageHandler("LOADED ACOUSTIC
SIMULATOR");
            }
            else
            {
                this.generalMessageHandler("FAILED TO LAOD
ACOUSTIC SIMULATOR");
            }
        }
        catch(ClassNotFoundException cnfe)
        {
            this.generalMessageHandler("ERROR: CLASS NOT
FOUND EXCEPTION WHEN TRYING TO LOAD ACOUSTIC MODEL.");
        }
        catch(InstantiationException ie)
        {
            this.generalMessageHandler("ERROR: INSTANTIATION
EXCEPTION WHEN LOADING ACOUSTIC SIM.");
        }
        catch(IllegalAccessException iae)
        {
            this.generalMessageHandler("ERROR: ILLEGAL
ACCESS EXCEPTION WHEN LOADING ACOUSTIC SIM.");
        }
    }
```

```
    private void
jMenuItem_StopNetManActionPerformed(java.awt.event.ActionEve
nt evt)
    {

this.controller.processUserInput(SimController.USERCOMMANDS.
STOPSERVER);
    }

    private void
jMenuItem_StartNetManActionPerformed(java.awt.event.ActionEv
ent evt)
    {

this.controller.processUserInput(SimController.USERCOMMANDS.
STARTSERVER);
    }

    private void
jMenuItem_SetPortNumActionPerformed(java.awt.event.ActionEve
nt evt)
    {
        int port_number;
        String input;
        while(true)
        {
            input = JOptionPane.showInputDialog(this,"Enter
a port number between 1 and 65535:","Enter
Port",JOptionPane.QUESTION_MESSAGE);
            if(input == null)
            {
                break;
            }
            else
            {
                try
                {
                    port_number = Integer.parseInt(input);
                    if(port_number <= 0 || port_number >=
65536)
                    {

JOptionPane.showMessageDialog(this,"Not a valid port
number.", "Invalid port
number",JOptionPane.WARNING_MESSAGE);
                    }
```

```
                    else
                    {

this.controller.processUserInput(SimController.USERCOMMANDS.
SETSERVERPORT,port_number);
                        break;
                    }
                }
                catch(NumberFormatException nfe)
                {
                    JOptionPane.showMessageDialog(this,"Not
a number.","Invalid port
number",JOptionPane.WARNING_MESSAGE);
                }
            }
        }

    }

    private void
jMenuItem_GetServerInfoActionPerformed(java.awt.event.Action
Event evt)
    {

this.controller.processUserInput(SimController.USERCOMMANDS.
SERVERINFO);
    }

    private void
formWindowClosing(java.awt.event.WindowEvent evt)
    {
        jMenuItem_ExitActionPerformed(null);
    }

    private void
jMenuItem_ExitActionPerformed(java.awt.event.ActionEvent
evt)
    {

this.controller.processUserInput(SimController.USERCOMMANDS.
STOPSIM);
        System.exit(0);
    }
```

```
    private void
jMenuItem_StopActionPerformed(java.awt.event.ActionEvent
evt)
    {

this.controller.processUserInput(SimController.USERCOMMANDS.
STOPSIM);
    }

    private void
jMenuItem_StartActionPerformed(java.awt.event.ActionEvent
evt)
    {


this.controller.processUserInput(SimController.USERCOMMANDS.
STARTSIM);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                new JFrameCommandGUI().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JEditorPane jEditorPane_InfoPane;
    private javax.swing.JMenu jFileMenu;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem_Exit;
    private javax.swing.JMenuItem jMenuItem_GetServerInfo;
    private javax.swing.JMenuItem jMenuItem_LoadAcousticSim;
    private javax.swing.JMenuItem jMenuItem_LoadLocationSim;
    private javax.swing.JMenuItem jMenuItem_SetPortNum;
    private javax.swing.JMenuItem jMenuItem_Start;
    private javax.swing.JMenuItem jMenuItem_StartNetMan;
    private javax.swing.JMenuItem jMenuItem_Stop;
    private javax.swing.JMenuItem jMenuItem_StopNetMan;
```

```
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JMenu jServerMenu;
    private javax.swing.JMenu jSimulatorMenu;
    // End of variables declaration

    public void errorMessageHandler(String msg)
    {
        this.sim_info += this.sdf.format(new
Date(System.currentTimeMillis())) + ">>> " + msg + "\n";
        this.jEditorPane_InfoPane.setText(this.sim_info);
    }

    public void generalMessageHandler(String msg)
    {
        this.sim_info += this.sdf.format(new
Date(System.currentTimeMillis())) + ">>> " + msg + "\n";
        this.jEditorPane_InfoPane.setText(this.sim_info);
    }

    public String getUserInput(String request, String...
choices)
    {
        return null;
    }

}
```

4. SphericalAcousticModel.java

```
/*
 * SphericalAcousticModel.java
 *
 * Created on August 1, 2007, 12:09 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package SimServer;

import Utilities.AcousticSignal;
import Utilities.Location;

/**
 *
 * @author richard betancourt
```

```
 */
public class SphericalAcousticModel implements IAcousticSim
{
    /**The sound speed in meters/second*/
    private static final int SOUND_SPEED = 1500;

    /**Load the shared library*/
    static
    {
        System.loadLibrary("sphericalmodel");
    }

    private native double calculate(double distance, double
src_power);

    /** Creates a new instance of SphericalAcousticModel */
    public SphericalAcousticModel()
    {
    }

    /**
     * Determine the acoustic signal properties.
     * @param source - The source node location
     * @param destination - The destination node location
     * @param as - The acoustic signal
     */
    public AcousticSignal processAcousticSignal(Location
source, Location destination, AcousticSignal as)
    {
       AcousticSignal new_signal = new AcousticSignal();

       /**Calculate the distance between the nodes*/
       double dist =
distance(source.getLatitude(),source.getLongitude(),
destination.getLatitude(), destination.getLongitude());
       double new_power = calculate(dist,
(double)as.getPower());
       new_signal.setPower((int)new_power);
       new_signal.setFrequency(as.getFrequency());
       new_signal.setData(as.getData());
       new_signal.setTransTime(as.getTransTime());

       long delay = (long)(dist/((double)SOUND_SPEED/1000));

       new_signal.setDelay(delay);
```

```
       return new_signal;


    }

    /**
     * For this acoustic model, this method performs no
function
     */
    public void startAcousticSim()
    {

    }

    /**
     * For this acoustic model, this method performs no
function
     */
    public void stopAcousticSim()
    {
    }

    /**
     * Calculates distance in meters between two points
given as latitude and
     * longitude in decimal degree format.
     * @author Brian Long
     * @param lat1 the latitude of the first node
     * @param lon1 the longitude of the first node
     * @param lat2 the latitude of the second node
     * @param lon2 the longitude of the second node
     * @return a <code>double</code> representation of the
distance between nodes in meters.
     */
    private static double distance(double lat1, double lon1,
double lat2, double lon2)
    {
      double theta = lon1 - lon2;
      double dist = Math.sin(Math.toRadians(lat1)) *
Math.sin(Math.toRadians(lat2)) +
                    Math.cos(Math.toRadians(lat1)) *
Math.cos(Math.toRadians(lat2)) *
                    Math.cos(Math.toRadians(theta));
      dist = Math.toDegrees(Math.acos(dist));
      dist = dist * 60; //nautical miles
      dist = dist * 1852; //meters
```

```
      return dist;
    }//end distance



}
```

5. sphericalmodel.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class SimServer_SphericalAcousticModel */

#ifndef _Included_SimServer_SphericalAcousticModel
#define _Included_SimServer_SphericalAcousticModel
#ifdef __cplusplus
extern "C" {
#endif
#undef SimServer_SphericalAcousticModel_SOUND_SPEED
#define SimServer_SphericalAcousticModel_SOUND_SPEED 1500L
/*
 * Class:     SimServer_SphericalAcousticModel
 * Method:    calculate
 * Signature: (DD)D
 */
JNIEXPORT jdouble JNICALL
Java_SimServer_SphericalAcousticModel_calculate
  (JNIEnv *, jobject, jdouble, jdouble);

#ifdef __cplusplus
}
#endif
#endif
```

6. sphericalmodel.c

```
#include <jni.h>

#include "stdafx.h"
#include <math.h>
#include "sphericalmodel.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```
BEGIN_MESSAGE_MAP(CsphericalmodelApp, CWinApp)
END_MESSAGE_MAP()


// CsphericalmodelApp construction

CsphericalmodelApp::CsphericalmodelApp()
{
     // TODO: add construction code here,
     // Place all significant initialization in
InitInstance
}


// The one and only CsphericalmodelApp object

CsphericalmodelApp theApp;


// CsphericalmodelApp initialization

BOOL CsphericalmodelApp::InitInstance()
{
     CWinApp::InitInstance();

     return TRUE;
}

JNIEXPORT jdouble JNICALL
Java_SimServer_SphericalAcousticModel_calculate
  (JNIEnv *env, jobject obj, jdouble distance, jdouble
src_pwr)
{
     jdouble transloss = 20.0 * log10(src_pwr);

     return src_pwr - transloss;

}
```

B. SIMULATION CLIENT SOURCE CODE

1. SimClientController.java

```
package SimClient;

import Utilities.ICommandUI;
import Utilities.Location;
import Utilities.ISimController;
import Utilities.ISimNetworkManager;
import Utilities.SimCommsThread;
import Utilities.SimInternalMessage;
import java.util.Enumeration;
import java.util.NoSuchElementException;
import java.util.Hashtable;
import Utilities.AcousticSignal;
import Utilities.SimMessage;
import Utilities.UANSimException;
import java.util.LinkedList;
import java.util.Random;
import java.io.IOException;
import java.lang.Math;

/**
 *
 * @author richard betancourt
 */
public class SimClientController implements ISimController
{
    private static final int MAXRAND =
(int)Math.pow(2.0,31.0);

    private ICommandUI cui;

    private ISimNetworkManager snm;

    private LinkedList<SimInternalMessage> comms_inputqueue;

    private LinkedList<SimInternalMessage> admin_inputqueue;

    private LinkedList<SimInternalMessage> outputqueue;

    private ClientInputQueueHandler iqh;

    private ClientOutputQueueHandler oqh;
```

```
    private Random rand;

    private String clientName;

    private BasicPingStack basicstack;

    private Location init_location;

    /** Creates a new instance of SimClientController */
    public SimClientController(ICommandUI ui,
ISimNetworkManager snm)
    {
        this.cui = ui;
        this.snm = snm;
        this.comms_inputqueue = new
LinkedList<SimInternalMessage>();
        this.admin_inputqueue = new
LinkedList<SimInternalMessage>();
        this.outputqueue = new
LinkedList<SimInternalMessage>();
        this.iqh = new
ClientInputQueueHandler(this.comms_inputqueue,
this.admin_inputqueue,this);
        this.basicstack = new
BasicPingStack(this,this.cui,this.snm);

((JFramClientCommandGUI)this.cui).setProtocolStack(basicstac
k);
        //IProtocolStack ps =
((JFramClientCommandGUI)ui).getBasicPingStack();
        //if(ps == null)
        //    System.out.println("PS NULL AFTER GET PROTOCOL
STACK");
        this.iqh.setProtocolStack(basicstack);
        this.oqh = new
ClientOutputQueueHandler(this.snm,this.outputqueue);

        this.oqh.start();
        this.iqh.start();


    }

    public void processInComms(SimInternalMessage sim)
    {
        synchronized(this.comms_inputqueue)
```

```
        {
            this.comms_inputqueue.add(sim);
            this.iqh.interrupt();
        }
    }

    public void processOutComms(SimInternalMessage sim)
    {
        System.out.println("IN PROCESSOUTCOMMS");
        synchronized(this.outputqueue)
        {
            this.outputqueue.add(sim);
            this.oqh.interrupt();
        }
    }

    public void processAction(SimInternalMessage sim)
    {
        //Should never get this from the server on the
client side
    }

    public void processAdmin(SimInternalMessage sim)
    {
        if(((String)sim.getContent()).equals("NAME_OK"))
        {
            this.cui.generalMessageHandler("CONNECTED TO
SERVER");

            //SEND LOCATION INFORMATION
            SimMessage msg = new SimInternalMessage();

msg.setMessageType(SimMessage.MESSAGETYPE.ACTION);
            msg.setContent(init_location);

((SimClientNetworkManager)this.snm).sendMessage(msg);
        }
        else
if(((String)sim.getContent()).equals("LOCATION_OK"))
        {
            //Don't need to do anything with this
            this.cui.generalMessageHandler("LOCATION SET");
        }
        else
if(((String)sim.getContent()).equals("DISCONNECT_OK"))
        {
```

```
            this.cui.generalMessageHandler("DISCONNECTED
FROM SERVER");
        }
        else
if(((String)sim.getContent()).equals("SIM_START"))
        {
            this.cui.generalMessageHandler("SIMULATION
STARTED");
        }
        else
if(((String)sim.getContent()).equals("SIM_STOP"))
        {
            this.cui.generalMessageHandler("SIMULATION
STOPPED");
        }
        else
        {
            this.cui.errorMessageHandler("UNKNOWN SIM
MESSAGE: "+((String)sim.getContent()));
        }
    }

    public void processDataToNode(SimInternalMessage sim)
    {
        //Do not need to do anything here.
    }

    public String processUserRequest(String request, String
... choices)
    {
       return null;
    }

    public void processFatalError(String msg)
    {
        this.cui.errorMessageHandler(msg);
    }

    public void processError(String msg)
    {
        this.cui.errorMessageHandler(msg);
    }

    public void setClientName(String name)
    {
        this.clientName = name;
```

```
    }

    public String getClientName()
    {
        return this.clientName;
    }

    public void setLocation(Location loc)
    {
        this.init_location = loc;
    }

    public Location getLocation()
    {
        return this.init_location;
    }

}
```

2. SimClientNetworkManager.java

```
package SimClient;

import Utilities.*;
import java.io.*;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.util.LinkedList;

/**
 *
 * @author richard betancourt
 */
public class SimClientNetworkManager implements
ISimNetworkManager
{
    /**Private data members*/
    private static final int DEFAULT_SERVER_PORT = 2875;

    private static final int DEFAULT_CLIENT_PORT = 2876;

    /**The client's port number*/
    private int client_port_num;

    /**The server's port number*/
```

```
    private int server_port_num;

    /**The server's ip address*/
    private InetAddress server_ip_address;

    /**A reference to the client's controller object*/
    private ISimController controller;

    /**Boolean flag to note the current status of the client
network manager*/
    private boolean running;

    /**Boolean flag to note when we need to stop the client
network manager*/
    private boolean keepRunning;

    /**Socket to be used for communicating with the server*/
    private Socket socket;

    /**The communications thread that will be used to send
and receive messages from the server*/
    private SimCommsThread sct;



    /** Creates a new instance of SimClientNetworkManager */
    public SimClientNetworkManager(int port, ISimController
c)
    {
        this.client_port_num = port;
        this.controller = c;
        this.running = false;

        this.server_ip_address = null;
        this.server_port_num = this.DEFAULT_SERVER_PORT;
        this.sct = null;
        this.socket = null;
    }

    public SimClientNetworkManager()
    {
        this.client_port_num = this.DEFAULT_CLIENT_PORT;
        this.controller = null;
        this.running = false;

        this.server_ip_address = null;
```

```
        this.server_port_num = this.DEFAULT_SERVER_PORT;
        this.sct = null;
        this.socket = null;
    }

    public void setController(ISimController sc)
    {
        this.controller = sc;
    }

    /**
     * Set the port number that this client will use to
communicate to the server
     * with.
     * @param portNumber The port number to set the client
socket to
     *
     */
    public void setPort(int portNumber)
    {
        this.client_port_num = portNumber;

    }

    /**
     * Set the port number of the server that this client
will communicate with.
     * @param portNumber The port number of the server
     */
    public void setServerPort(int portNumber)
    {
        this.server_port_num = portNumber;
    }

    /**
     * Set the IP address of the server.
     * @param ipadd The IP address of the server.
     */
    public void setServerIPAddress(InetAddress ipadd)
    {
        this.server_ip_address = ipadd;
    }


    public int getServerPort()
    {
```

```
        return this.server_port_num;
    }

    public InetAddress getServerIPAddress()
    {
        return this.server_ip_address;
    }

    /**
     * Start the client network manager.
     */
    public void startNetworkManager()
    {
        //CREATE THE SOCKET TO THE SERVER
        try
        {
            this.socket = new Socket();
            this.socket.setKeepAlive(true);
            this.socket.connect(new
InetSocketAddress(this.server_ip_address,
this.server_port_num));

        }
        catch(IOException ioe)
        {
            this.controller.processError("UNABLE TO CONNECT
TO SERVER AT: "+this.server_ip_address.toString()+" ON PORT:
"+this.server_port_num);
        }

        //INSTANTIATE THE SIMCOMMSTHREAD
        this.sct = new SimCommsThread(this.socket,
((SimClientController)this.controller).getClientName(),
this.controller);
        this.sct.start();

        //pass the server our name and location information
        SimInternalMessage msg = new SimInternalMessage();
        msg.setMessageType(SimMessage.MESSAGETYPE.ADMIN);
        msg.setContent("CLIENT_NAME
"+((SimClientController)this.controller).getClientName());
        this.controller.processOutComms(msg);

        this.running = true;

        System.out.println("SENDING CLIENT NAME");
```

```
    }

    /**
     * Stop the network manager.
     */
    public void stopNetworkManager()
    {
        if(this.sct == null)
        {
            return;
        }
        //tell the server we are disconnecting
        SimMessage msg = new SimMessage();
        msg.setMessageType(SimMessage.MESSAGETYPE.ADMIN);
        msg.setContent("DISCONNECT");

        this.sct.sendMessage(msg);

        while(this.socket.isConnected())
        {

        }

        this.running = false;
        this.sct = null;
    }

    public boolean isRunning()
    {
        return this.running;
    }

    public int getPort()
    {
        return this.client_port_num;
    }

    public String getIPAddress()
    {
        if(this.socket.isConnected())
        {
            String ipaddress =
this.socket.getInetAddress().toString();
            int index = ipaddress.indexOf("/");
            if(index == -1)
                return ipaddress;
```

```
            else
                return ipaddress.substring(index+1);
        }

        return null;
    }


    /**
     * Send a message out into the simulated environment.
     * @param sm The message packet we will send out
     */
    public void sendMessage(SimMessage sm)
    {
        if(sm == null)
            System.out.println("NULL MESSAGE");
        else
            this.sct.sendMessage(sm);
    }



}
```

3. JFramClinetCommandGUI.java

```
package SimClient;

import Utilities.*;
import Utilities.ICommandUI;
import java.io.File;
import java.net.*;
import java.util.Calendar;
import java.util.Date;
import javax.swing.JOptionPane;
import java.text.SimpleDateFormat;

/**
 *
 * @author  richard betancourt
 */
public class JFramClientCommandGUI extends
javax.swing.JFrame implements ICommandUI
{
```

```
    private static final String GREETING = "UNDERWATER
ACOUSTIC NETWORK CLIENT SIMULATOR, VERSION 0.2\nDEVELOPED BY
RICHARD BETANCOURT AND BRIAN LONG\n\n";

    private static final String DATE_FORMAT = "yyyy-MM-dd
HH:mm:ss";

    private static SimpleDateFormat sdf;

    /**The main simulation controller*/
    private SimClientController controller;

    private IProtocolStack bps;

    private SimClientNetworkManager scnm;

    /**A string which will be used to hold information
messages that are coming
     * back from the simulation controller and given to the
jEditorPane to display
     * for the users.
     */
    private String sim_info;

    /** Creates new form JFramClientCommandGUI */
    public JFramClientCommandGUI()
    {
        initComponents();

        /**Initialized the simulation controller*/
        this.scnm = new SimClientNetworkManager();

        this.controller = new SimClientController(this,
this.scnm);

        this.scnm.setController(this.controller);

        //this.bps = null;
        //= new BasicPingStack(controller,this,this.scnm);
        //if(this.bps == null)
        //    System.out.println("BPS NULL IN CONSTRUCTOR");

        sim_info = this.GREETING;

        this.jEditorPane_InfoPane.setText(sim_info);
```

```
        this.sdf = new SimpleDateFormat(DATE_FORMAT);

    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="
Generated Code ">
    private void initComponents()
    {
        jScrollPane1 = new javax.swing.JScrollPane();
        jEditorPane_InfoPane = new
javax.swing.JEditorPane();
        jMenuBar1 = new javax.swing.JMenuBar();
        jFileMenu = new javax.swing.JMenu();
        jMenuItem_Exit = new javax.swing.JMenuItem();
        jClientMenu = new javax.swing.JMenu();
        jMenuItem_SetClientName = new
javax.swing.JMenuItem();
        jMenuItem_ClientPort = new javax.swing.JMenuItem();
        jMenuItem_ServerPort = new javax.swing.JMenuItem();
        jMenuItem_ServerIP = new javax.swing.JMenuItem();
        jMenuItem_Connect = new javax.swing.JMenuItem();
        jMenuItem_Disconnect = new javax.swing.JMenuItem();
        jMenuItem_SetLocation = new javax.swing.JMenuItem();
        jPingMenu = new javax.swing.JMenu();
        jMenuItem_SendPing = new javax.swing.JMenuItem();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON
_CLOSE);
        setTitle("UAN Sim Client");
        jEditorPane_InfoPane.setEditable(false);
        jScrollPane1.setViewportView(jEditorPane_InfoPane);

        jFileMenu.setText("File");
        jMenuItem_Exit.setText("Exit");
        jMenuItem_Exit.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
```

```
            {
                jMenuItem_ExitActionPerformed(evt);
            }
        });

        jFileMenu.add(jMenuItem_Exit);

        jMenuBar1.add(jFileMenu);

        jClientMenu.setText("Client");
        jMenuItem_SetClientName.setText("Set Client Name");
        jMenuItem_SetClientName.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_SetClientNameActionPerformed(evt);
            }
        });

        jClientMenu.add(jMenuItem_SetClientName);

        jMenuItem_ClientPort.setText("Set Client Port");
        jMenuItem_ClientPort.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_ClientPortActionPerformed(evt);
            }
        });

        jClientMenu.add(jMenuItem_ClientPort);

        jMenuItem_ServerPort.setText("Set Server Port");
        jMenuItem_ServerPort.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_ServerPortActionPerformed(evt);
            }
        });
```

```
        jClientMenu.add(jMenuItem_ServerPort);

        jMenuItem_ServerIP.setText("Set Server IP Address");
        jMenuItem_ServerIP.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_ServerIPActionPerformed(evt);
            }
        });

        jClientMenu.add(jMenuItem_ServerIP);

        jMenuItem_Connect.setText("Connect to Server");
        jMenuItem_Connect.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_ConnectActionPerformed(evt);
            }
        });

        jClientMenu.add(jMenuItem_Connect);

        jMenuItem_Disconnect.setText("Disconnect From
Server");
        jMenuItem_Disconnect.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_DisconnectActionPerformed(evt);
            }
        });

        jClientMenu.add(jMenuItem_Disconnect);

        jMenuItem_SetLocation.setText("Set Client
Location");
```

```
        jMenuItem_SetLocation.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_SetLocationActionPerformed(evt);
            }
        });

        jClientMenu.add(jMenuItem_SetLocation);

        jMenuBar1.add(jClientMenu);

        jPingMenu.setText("Ping App");
        jMenuItem_SendPing.setText("Send Ping");
        jMenuItem_SendPing.addActionListener(new
java.awt.event.ActionListener()
        {
            public void
actionPerformed(java.awt.event.ActionEvent evt)
            {
                jMenuItem_SendPingActionPerformed(evt);
            }
        });

        jPingMenu.add(jMenuItem_SendPing);

        jMenuBar1.add(jPingMenu);

        setJMenuBar(jMenuBar1);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
            .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 400, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
```

```
            .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 279, Short.MAX_VALUE)
        );
        pack();
    }// </editor-fold>

    private void
jMenuItem_SetLocationActionPerformed(java.awt.event.ActionEv
ent evt)
    {
        String input;
        Location loc;
        input = JOptionPane.showInputDialog(this,"Enter
location
information","Location",JOptionPane.QUESTION_MESSAGE);
        while(true)
        {
            try
            {
                String[] inputarray = input.split(",");
                loc = new Location();

loc.setLatitude(Double.parseDouble(inputarray[0]));

loc.setLongitude(Double.parseDouble(inputarray[1]));

loc.setDepth(Integer.parseInt(inputarray[2]));
                break;
            }
            catch(LocationException le)
            {
                JOptionPane.showMessageDialog(this,"INVALID
INPUT");
            }
        }
        this.controller.setLocation(loc);
    }

    private void
jMenuItem_SetClientNameActionPerformed(java.awt.event.Action
Event evt)
    {
        String input;
        input = JOptionPane.showInputDialog(this,"Enter a
name for this client","Client
Name",JOptionPane.QUESTION_MESSAGE);
```

```
        this.controller.setClientName(input);
    }

    private void
jMenuItem_SendPingActionPerformed(java.awt.event.ActionEvent
evt)
    {
        String input;
        input = JOptionPane.showInputDialog(this,"Enter ID
of Node to Ping","Node ID",JOptionPane.QUESTION_MESSAGE);
        if(input == null)
            return;
        this.bps.processOutputAcousticSignal(input);

    }

    private void
jMenuItem_DisconnectActionPerformed(java.awt.event.ActionEve
nt evt)
    {
        this.scnm.stopNetworkManager();
    }

    private void
jMenuItem_ConnectActionPerformed(java.awt.event.ActionEvent
evt)
    {
        this.scnm.startNetworkManager();
    }

    private void
jMenuItem_ServerIPActionPerformed(java.awt.event.ActionEvent
evt)
    {
        InetAddress ipaddress;
        String input;
        boolean running = true;
        while(running)
        {
            input = JOptionPane.showInputDialog(this,"Enter
IP Address of Server","Enter IP
Address",JOptionPane.QUESTION_MESSAGE);
            if(input == null)
            {
                break;
            }
```

```
            else
            {
                try
                {
                    ipaddress =
InetAddress.getByName(input);

                    this.scnm.setServerIPAddress(ipaddress);
                    running = false;

                }
                catch(UnknownHostException uhe)
                {

JOptionPane.showMessageDialog(this,"Invalid IP
Address","Invalid IP Address",JOptionPane.WARNING_MESSAGE);
                }
            }
        }
    }

    private void
jMenuItem_ServerPortActionPerformed(java.awt.event.ActionEve
nt evt)
    {
        int port_number;
        String input;
        while(true)
        {
            input = JOptionPane.showInputDialog(this,"Enter
a port number between 1 and 65535:","Enter
Port",JOptionPane.QUESTION_MESSAGE);
            if(input == null)
            {
                break;
            }
            else
            {
                try
                {
                    port_number = Integer.parseInt(input);
                    if(port_number <= 0 || port_number >=
65536)
                    {

JOptionPane.showMessageDialog(this,"Not a valid port
```

```
number.", "Invalid port
number",JOptionPane.WARNING_MESSAGE);
                    }
                    else
                    {

this.scnm.setServerPort(port_number);
                        break;
                    }
                }
                catch(NumberFormatException nfe)
                {
                    JOptionPane.showMessageDialog(this,"Not
a number.","Invalid port
number",JOptionPane.WARNING_MESSAGE);
                }
            }
        }
    }

    private void
jMenuItem_ClientPortActionPerformed(java.awt.event.ActionEve
nt evt)
    {
        int port_number;
        String input;
        while(true)
        {
            input = JOptionPane.showInputDialog(this,"Enter
a port number between 1 and 65535:","Enter
Port",JOptionPane.QUESTION_MESSAGE);
            if(input == null)
            {
                break;
            }
            else
            {
                try
                {
                    port_number = Integer.parseInt(input);
                    if(port_number <= 0 || port_number >=
65536)
                    {

JOptionPane.showMessageDialog(this,"Not a valid port
```

```
number.", "Invalid port
number",JOptionPane.WARNING_MESSAGE);
                    }
                    else
                    {
                        this.scnm.setPort(port_number);
                        break;
                    }
                }
                catch(NumberFormatException nfe)
                {
                    JOptionPane.showMessageDialog(this,"Not
a number.","Invalid port
number",JOptionPane.WARNING_MESSAGE);
                }
            }
        }
    }

    private void
jMenuItem_ExitActionPerformed(java.awt.event.ActionEvent
evt)
    {
        this.scnm.stopNetworkManager();
        System.exit(0);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                new
JFramClientCommandGUI().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JMenu jClientMenu;
    private javax.swing.JEditorPane jEditorPane_InfoPane;
    private javax.swing.JMenu jFileMenu;
```

```
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem_ClientPort;
    private javax.swing.JMenuItem jMenuItem_Connect;
    private javax.swing.JMenuItem jMenuItem_Disconnect;
    private javax.swing.JMenuItem jMenuItem_Exit;
    private javax.swing.JMenuItem jMenuItem_SendPing;
    private javax.swing.JMenuItem jMenuItem_ServerIP;
    private javax.swing.JMenuItem jMenuItem_ServerPort;
    private javax.swing.JMenuItem jMenuItem_SetClientName;
    private javax.swing.JMenuItem jMenuItem_SetLocation;
    private javax.swing.JMenu jPingMenu;
    private javax.swing.JScrollPane jScrollPane1;
    // End of variables declaration


    public void errorMessageHandler(String msg)
    {
        this.sim_info += this.sdf.format(new
Date(System.currentTimeMillis())) + ">>> " + msg + "\n";
        this.jEditorPane_InfoPane.setText(this.sim_info);
    }

    public void generalMessageHandler(String msg)
    {
        this.sim_info += this.sdf.format(new
Date(System.currentTimeMillis())) + ">>> " + msg + "\n";
        this.jEditorPane_InfoPane.setText(this.sim_info);
    }

    public String getUserInput(String request, String...
choices)
    {
        return null;
    }

    public void setProtocolStack(IProtocolStack ips)
    {
        this.bps = ips;
    }


}
```

4. BasicPingStack.java

```
package SimClient;
```

```
import Utilities.*;
import java.net.*;
import java.io.*;

/**
 *
 * @author richard betancourt
 */
public class BasicPingStack extends Thread implements
IProtocolStack
{

    private ISimController controller;

    private ICommandUI cui;

    private ISimNetworkManager netmngr;

    private CollisionDetection cd;

    /** Creates a new instance of BasicPingStack */
    public BasicPingStack(ISimController cont, ICommandUI c,
ISimNetworkManager n)
    {
        this.controller = cont;
        this.cui = c;
        this.netmngr = n;
        this.cd = new CollisionDetection(this.cui);
        this.cd.setProtocolStack(this);
        this.cd.start();

    }

    public void processInputAcousticSignal(AcousticSignal
as)
    {
        if(this.cd.setAcousticSignal(as))
        {
           cd.interrupt();
        }
    }

    public void processOutputAcousticSignal(Object o)
    {
        processLevel1Out((String)o);
```

```
    }

    public void processLevel1In(AcousticSignal as)
    {
        System.out.println("IN PROCESS LEVEL 1 IN");
        //TODO: PROCESS PING
        //For the purposes of this small demo protocol
stack, level one processing does nothing
        //except report that a ping was received to the
user.
        this.cui.generalMessageHandler("PING RECEIVED FROM:
"+as.getData().getSrcName());
    }

    public void processLevel1Out(String dest_name)
    {
        Packet p = new Packet(Packet.DATA,null);

p.setSrcName(((SimClientController)this.controller).getClien
tName());
        AcousticSignal newAs = new AcousticSignal();
        newAs.setData(p);
        newAs.setFrequency(33.0);
        newAs.setPower(250);
        newAs.setTransTime(System.currentTimeMillis());
        SimInternalMessage sim = new SimInternalMessage();
        sim.setContent(newAs);
        sim.setMessageType(SimMessage.MESSAGETYPE.COMMS);
        this.controller.processOutComms(sim);
    }



}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] F. Akyildiz, D. Pompili, and T. Melodia. *Underwater Acoustic Sensor Networks: Research Challenges*. Ad Hoc Networks Journal (Elseviier), pp. 257-279, March 2005.

[2] Jose Coelho, "Underwater Acoustic Networks: Evaluation of the Impact of Media Access Control on Latency, in a Delay Constrained Network," Master's Thesis (MS-CS), Naval Postgraduate School, Monterey, California, March 2005.

[3] Brian Long, "Implementation of a Distributed Time Based Simulation of Underwater Acoustic Networking Using Java," Master's Thesis (MS-CS), Naval Postgraduate School, Monterey, California, September 2006.

[4] J. G. Proakis, E. M. Sozer, J. A. Rice, and M. Stojanovic, *Shallow Water Acoustic Networks*, IEEE Communication Magazine, Vol. 39, No. 11, pp. 114-119, November 2001.

[5] J. G. Proakis, E. M. Sozer, and M. Stojanovic, *Underwater Acoustic Networks*, IEEE Journal of Oceanic Engineering, Vol. 25, No. 1, pp. 72-83, January 2000.

[6] S. G. Chappell, R. J. Komerska, *A Simulation Environment for Testing and Evaluating Multiple Cooperating Solar-Powered AUVs*, Proceeding of the MTS/IEEE Oceans 2006 Conference, September 2006.

[7] S. G. Chappell, R. J. Komerska, *An Environment for High-Level Multiple AUV Simulation and Communication*, http://www.ausi.org/publications/clout2000.pdf, April 2007.

[8] The Navy Unmanned Undersea Vehicle (UUV) Master Plan, November 2004, http://www.navy.mil/navydata/technology/uuvmp.pdf, April 2007.

[9] Paul C. Etter, Underwater Acoustic Modeling and Simulation, 3rd Ed., Spon Press, New York, NY. 2003.

[10] Alan B. Coppens, Austin R. Frey, Lawrence E. Kinsler, James V. Sanders, Fundamentals of Acoustics, 4th Ed. John Wiley & Sons, New York, NY. 2000.

[11] Andrew E. Kramer, "Russia Set to Plant Flag on Artic Seabed." New York Times, August 2, 2007.

[12] J. G. Proakis, E. M. Sozer, and M. Stojanovic, *Initialization and Routing Optimization for Adhoc Underwater Acoustic Networks*, Proceeding of Opnetwork 2000.

[13] The Ocean Acoustics Library, http://oalib.hlsresearch.com/, April 2007.

[14] J. G. Proakis, E. M. Sozer, and M. Stojanovic, *Design and Simulation of an Underwater Acoustic Local Area Network*, Proceeding of Opnetwork 1999.

[15] Geoffrey Xie, John Gibson, Leopoldo Diaz-Gonzalez, *Incorporating Realistic Acoustic Propagation Models in Simulation of Underwater Acoustic Networks: A Statistical Approach*, MTS/IEEE Oceans Conference, September 2006.

[16] L. J. Solorzano, "Underwater Acoustic Networks: An Acoustic Propagation Model for Simulation of Underwater Acoustic Networks," Masters Thesis (MS-CS), Naval Postgraduate School, Monterey, California, December 2005.

[17] L. Prechelt, *An Empirical Comparison of Seven Programming Languages*, Computer, Vol. 33, No. 10, pp. 23-29, October 2000.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California